راهنمای عملی

مبانی برنامه‌نویسی به زبان جاوا

مونیکا پاولان

ترجمه‌ی قاسم کیانی

ghasemkiani@yahoo.com

نگارش ۰٫۰۱-۲۰۰۴۰۹۰۵

نام کتاب: راهنمای عملی مبانی برنامه‌نویسی به زبان جاوا
عنوان اصلی: Essentials of the Java Programming Language – A Hands-On Guide (Part 1)
نویسنده: مونیکا پاولان (Monica Pawlan)
مترجم: دکتر قاسم کیانی مقدم
ناشر:
شمارگان:
نوبت چاپ:
قیمت:
شابک:
منبع *http://developer.java.sun.com/developer/onlineTraining/Downloads/BasicJava1.zip*
http://developer.java.sun.com/developer/onlineTraining/Downloads/BasicJava2.zip
ISBN: 0201707209
ایمیل مترجم: *ghasemkiani@yahoo.com*

فهرست

پیشگفتار

این کتاب حدود پنج سال قبل نوشته شده و از ترجمه‌ی آن هم بیش از دو سال می‌گذرد. از این رو، نمی‌توان آن را اثر جدیدی در زمینه‌ی زبان برنامه‌نویسی جاوا دانست. اما به علت اینکه بیشتر به نکات پایه و اساسی زبان پرداخته، هنوز هم به خوبی قابل استفاده است، به ویژه که به عنوان یک کتاب الکترونیک رایگان در اختیار خوانندگان محترم قرار می‌گیرد. امید است خوانندگانی که زحمت خواندن کتاب را به خود می‌دهند، نظرات و انتقادات ارزشمند خود را از مترجم دریغ نکنند.

دکتر قاسم کیانی مقدم
ghasemkiani@yahoo.com
۱۳۸۳/۰۶/۱۶

مقدمه

در صورتی که شما آشنایی قبلی با یک زبان برنامه‌نویسی داشته و با مسایلی مانند نمایش متن یا گرافیک آشنا هستید، می‌توانید برای فرا گرفتن جاوا از این خودآموز بهره بگیرید. در این خودآموز یاد خواهید گرفت که چگونه از نرم‌افزار بستر جاوا ۲[1] برای ایجاد و اجرای سه نوع معمول برنامه‌هایی که به زبان جاوا نوشته می‌شوند، استفاده کنید: برنامه[2]، برنامک[3]، و سرولت[4].

تشابهات و اختلافات برنامه‌ها، برنامک‌ها، و سرولت‌ها، چگونگی ایجاد یک رابط کاربر اساسی که ورودی ساده‌ی کاربر نهایی را پردازش می‌کند، طریقه‌ی خواندن داده‌ها از پرونده یا پایگاه داده‌ای و نوشتن داده‌ها در آنها، و نحوه‌ی فرستادن و دریافت داده‌ها بر روی شبکه را خواهید آموخت. این خودآموز یک اثر جامع نیست، بلکه سعی دارد شما را از راهی هموار و مستقیم به سوی درک اصول کلی برنامه‌نویسی در بستر جاوا ببرد.

اگر هیچگونه زمینه‌ی قبلی در برنامه‌نویسی نداشته باشید، ممکن است باز هم بتوانید از این خودآموز بهره ببرید؛ ولی شاید بهتر باشد که قبلاً کتابی مانند خودآموز جاوا ۲ را بخوانید.

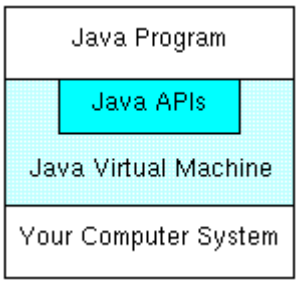
[1] Java® 2 Platform.
[2] application.
[3] applet.
[4] servlet.

فصل ۱: تدوین و اجرای یک برنامه‌ی ساده

عصر رایانه آغاز شده است. در خانه‌ها و شرکت‌ها در همه جای جهان از رایانه به طرق مختلف استفاده می‌شود، زیرا رایانه امکان انجام کارها را با سرعت، دقت، و کارآیی بالا در اختیار افراد قرار می‌دهد. رایانه کارهای مختلفی می‌تواند انجام دهد، از نمایش گرافیک متحرک سه‌بعدی همراه با صدای زمینه گرفته، تا محاسبه‌ی تعداد روزهای تعطیل شما، و یا پردازش لیست حقوقی کارکنان یک شرکت بزرگ.
هنگامی که می‌خواهید رایانه کاری برای شما انجام دهد، باید برایش برنامه بنویسید. برنامه مجموعه‌ای از دستورالعمل‌ها است که برای رایانه کار مورد نظر را مشخص می‌کند. در این فصل چگونگی نوشتن، تدوین کردن، و اجرا کردن برنامه‌ی ساده‌ای را به زبان جاوا فرا خواهید گرفت که یک رشته‌ی یک‌خطی را روی صفحه‌ی نمایشگر می‌نویسد.
اما قبل از آنکه بتوانید برنامه بنویسید و تدوین کنید، باید بفهمید که بستر جاوا چیست، و رایانه‌ی خود را برای اجرای برنامه‌ها آماده کنید.

چند کلمه در باره‌ی بستر جاوا

بستر جاوا متشکل از رابط‌های برنامه‌نویسی کاربردی[۱] (API) جاوا و دستگاه مجازی جاوا[۲] (JVM) است.



شکل ۱. بستر جاوا

APIهای جاوا کتابخانه‌هایی از کد تدوین شده هستند که می‌توانید از آنها در برنامه‌های خود استفاده کنید. این کتابخانه‌ها برخی از کارها را به طور آماده برای برنامه‌ی شما امکان‌پذیر می‌کنند، و تا حد زیادی از وقت برنامه‌نویسی شما کم می‌کنند.
برنامه‌ی جاوای ساده‌ای که در این فصل خواهیم نوشت، با استفاده از API جاوا یک متن یک‌سطری را روی نمایشگر چاپ می‌کند. توانایی چاپ کردن روی نمایشگر توسط API به طور آماده در اختیار ما قرار گرفته است؛ فقط کافی است که ما متن مورد نظر را مشخص کنیم.
برنامه‌های جاوا به وسیله‌ی برنامه‌ی دیگری به نام JVM اجرا (یا تفسیر[۳]) می‌شوند. در صورتی که با بیسیک بصری[۱] یا زبان‌های تفسیر شده‌ی دیگر آشنا باشید، احتمالاً این مفهوم برای شما تازگی نخواهد داشت. برنامه

[1] application programming interface.
[2] Java virtual machine.
[3] interpret.

به جای اینکه مستقیماً به وسیله‌ی سامانه‌ی عامل رایانه اجرا شود، توسط JVM برای سامانه‌ی عامل اجرا می‌شود. این بدان معنا است که هر سامانه‌ی رایانه‌ای که JVM روی آن نصب شده باشد، می‌تواند برنامه‌های جاوا را صرف نظر از اینکه روی چه سامانه‌ی عاملی ایجاد شده باشند، اجرا کند.
مثلاً برنامه‌ی جاوایی که روی یک رایانه‌ی شخصی با سامانه‌ی عامل ویندوز NT تهیه شده است، بدون هرگونه تغییری روی یک ایستگاه کاری اولترای سان[۲] با سامانه‌ی عامل سولاریس[۳] نیز اجرا خواهد شد، و بر عکس.

تنظیم کردن رایانه

پیش از آنکه بتواند برنامه‌ی ساده‌ی این فصل را بنویسید و اجرا کنید، لازم است که بستر جاوا را روی سامانه‌ی رایانه‌ای خود نصب کنید.
بستر جاوا را می‌توانید به رایگان از پایگاه رایانه‌ای *http://java.sun.com* بگیرید. در این پایگاه نرم‌افزار بستر جاوا برای ویندوز NT/۹۸/۹۵ و یا برای سولاریس در دسترس است. اطلاعات لازم برای نصب و پیکربندی بستر جاوا به منظور نوشتن و اجرا کردن برنامه‌های جاوا در صفحه‌ی فروگذاری[۴] درج شده است.

توجه: قبل از اینکه بخواهید برنامه‌ی ساده‌ای را که در اینجا ارائه خواهد شد، بنویسید و اجرا کنید، باید حتماً بستر جاوا را نصب و پیکربندی کرده باشید.

نوشتن یک برنامه

ساده‌ترین راه برای نوشتن یک برنامه، استفاده از یک ویرایشگر متنی[۵] است. بنا بر این، با استفاده از ویرایشگر متنی مورد علاقه‌ی خود یک پرونده‌ی متنی ایجاد کنید و متن زیر را در آن بنویسید. یادتان باشد که این پرونده را با نام `ExampleProgram.java` ضبط کنید. در برنامه‌های جاوا افتراق حروف کوچک و بزرگ اهمیت دارد، بنا بر این، هنگام تایپ کردن به کوچک و بزرگ بودن حروف توجه خاصی مبذول کنید:

```
//A Very Simple Example
class ExampleProgram {
  public static void main(String[] args){
    System.out.println("I'm a Simple Program");
  }
}
```

تدوین کردن برنامه

برای اینکه هر رایانه‌ی مجهز به JVM بتواند برنامه‌ی شما را تفسیر و اجرا کند، لازم است که برنامه‌ی شما به شکلی تبدیل شود که JVM آن را بفهمد. تدوین[۶] کردن یک برنامه‌ی جاوا به معنای این است که متنی را که شما می‌توانید بفهمید در یک پرونده‌ی متنی (که به آن متن برنامه[۷] می‌گویند) بنویسید، و آن را به بایت‌کد[۸] که دستورالعمل‌های مستقل از بستر برای JVM هستند، تبدیل کنید.

---

[۱] Visual Basic.
[۲] Sun Ultra workstation.
[۳] Solaris.
[۴] download.
[۵] text editor.
[۶] compile.
[۷] source code.
[۸] bytecode.

تدوینگر جاوا[1] در سطر فرمان سامانه‌های عامل یونیکس[2] و پوسته‌ی داس[3] به صورت زیر فرا خوانده می‌شود:

```
javac ExampleProgram.java
```

توجه: بخشی از فرایند پیکربندی بستر جاوا تعیین مسیر کلاس‌ها[4] است. برای تعیین مسیر کلاس‌ها یا باید از گزینه‌ی -classpath به همراه فرمان تدوینگر javac یا فرمان تفسیرگر java استفاده کنید، و یا اینکه متغیر محیطی CLASSPATH را تعیین نمایید. باید مسیر کلاس‌ها را به گونه‌ای تعیین کنید که نشان دهنده‌ی شاخه‌ای باشد که کلاس ExampleProgram در آن واقع شده است، تا اینکه تدوینگر و تفسیرگر بتوانند آن را پیدا کنند. برای اطلاعات بیشتر به نشانی *http://java.sun.com/products/jdk/1.2/docs/tooldocs/tools.html* در اینترنت مراجعه کنید.

تفسیر و اجرای برنامه

اگر برنامه‌ی شما با موفقیت به بایت‌کد جاوا تدوین شود، می‌توانید برنامه‌ها را روی هر دستگاه مجازی جاوا تفسیر و اجرا نمایید و یا برنامک‌های خود را در هر مرورگر اینترنت[5] که مجهز به دستگاه مجازی جاوا باشد، مانند نتاسکیپ[6] یا اینترنت اکسپلورر[7] تفسیر و اجرا کنید. تفسیر و اجرای برنامه‌ی جاوا به معنای فرا خوانی تفسیرگر بایت‌کد دستگاه مجازی جاوا است که بایت‌کدهای جاوا را به کدهای ماشین اختصاصی بستر تبدیل می‌کند، تا اینکه رایانه بتواند برنامه را بفهمد و اجرا کند.

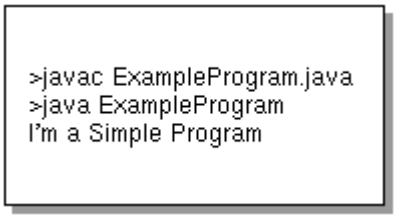
تفسیرگر جاوا از سطر فرمان سامانه‌های عامل یونیکس و پوسته‌ی داس به صورت زیر فرا خوانده می‌شود:

```
java ExampleProgram
```

در سطر فرمان باید نوشته‌ی زیر را ببینید:

```
I'm a Simple Program
```

در تصویر زیر کل جریان را در صفحه‌ی رایانه می‌بینید:



شکل ۲. برنامه‌ی نمونه

مشکلات شایع تدوینگر و تفسیرگر

در صورتی که در تدوین کردن و اجرای برنامه‌ی ساده‌ی این فصل با مشکل مواجه شدید، بهتر است به فصل "مشکلات شایع تدوینگر و تفسیرگر" در "خودآموز جاوا" به نشانی *http://java.sun.com/docs/books/tutorial/getStarted/problems/index.html* در اینترنت مراجعه کنید.

[1] the java compiler.
[2] Unix.
[3] DOS shell.
[4] class path.
[5] Web browser.
[6] Netscape.
[7] Internet Explorer.

توضیحات متن برنامه

در متن برنامه برای اینکه کسی که آن را می‌خواند بتواند از آن سر در بیاورد، توضیحاتی گنجانده می‌شود. گاهی هم در هنگام خطازدایی[۱] برای مشخص کردن منشأ یک خطا با گذاشتن علامت توضیح یک سطر را از جریان خارج می‌کنند. و نهایتاً کاربرد دیگر توضیحات برای تهیه‌ی مستندات[۲] API است. برای این مقاصد، سه نوع توضیح در جاوا تعبیه شده است: دو خط کج[۳]، به روش C، و توضیحات مستندسازی.

دو خط کج

از دو خط کج (//) در زبان C++ استفاده می‌شود، و به تدوینگر می‌گوید که از خط‌های کج تا آخر سطر همه چیز باید توضیح تلقی شود.

```
//A Very Simple Example
class ExampleProgram {
  public static void main(String[] args){
    System.out.println("I'm a Simple Program");
  }
}
```

به روش C

به جای دو خط کج، می‌توانید از توضیح به روش C (/* */) برای مشخص کردن یک توضیح یک یا چند سطری استفاده کنید.

```
/* These are
C-style comments
*/
class ExampleProgram {
  public static void main(String[] args){
    System.out.println("I'm a Simple Program");
  }
}
```

توضیحات مستندسازی

برای ایجاد مستندات برنامه‌ی خود، متن مورد نظر را در بین علامت توضیح مستندسازی (/** */) قرار دهید تا ابزار javadoc بتواند آن را پیدا کند. ابزار javadoc توضیحات مستندسازی قرار گرفتن در متن برنامه‌ها شناسایی می‌کند و با استفاده از آنها مستندات API را ایجاد می‌کند.

```
/** This class displays a text string at
*  the console.
*/
class ExampleProgram {
  public static void main(String[] args){
    System.out.println("I'm a Simple Program");
  }
}
```

وقتی فقط یک کلاس ساده داشته باشیم، دلیلی ندارد که مستندات API تهیه کنیم. این کار زمانی معنا پیدا می‌کند که برنامه‌ی شما متشکل از چندین کلاس پیچیده باشد که نیازمند مستندسازی هستند. ابزار

---
[۱] debugging.
[۲] documentation.
[۳] slash.

پیشگفته تعدادی صفحه‌ی HTML (صفحات اینترنت) ایجاد می‌کند که ساختار کلاس‌ها را توصیف می‌کنند، و متن توضیحات مستندسازی نیز در آنها گنجانده شده است. برای به دست آوردن اطلاعات بیشتر در باره‌ی فرمان javadoc و خروجی آن به صفحه‌ی اصلی javadoc به نشانی *http://java.sun.com/products/jdk/javadoc/index.html* مراجعه نمایید.

مستندسازی API

نصب بستر جاوا مشتمل بر مستندسازی API است، که APIهای موجود را که می‌توانید از آنها در برنامه‌های خود استفاده کنید، تشریح می‌کند. این پرونده‌ها در پوشه‌ای به نام doc زیر پوشه‌ای که بستر را نصب کرده‌اید، قرار دارند. مثلاً اگر بستر در پوشه‌ی /usr/local/java/jdk1.2 نصب شده باشد، مستندات API در پوشه‌ی /usr/local/java/jdk1.2/doc/api قرار خواهند داشت.

اطلاعات بیشتر

- برای اطلاعات بیشتر در باره‌ی تعیین مسیر کلاس‌ها و استفاده از فرمان‌های javac و java به صفحه‌ی "ابزارهای کیت برنامه‌نویسی[1] جاوا ۲" به نشانی *http://java.sun.com/products/jdk/1.2/docs/tooldocs/tools.html* مراجعه کنید.
- برای کمک در مورد مشکلاتی که با آن مواجه شده‌اید، می‌توانید به فصل "مشکلات شایع تدوینگر و تفسیرگر" در "خودآموز جاوا" به نشانی *http://java.sun.com/docs/books/tutorial/getStarted/problems/index.html* در اینترنت مراجعه کنید. نشانی خودآموز جاوا *http://java.sun.com/docs/books/tutorial/trailmap.html* است.
- همچنین، برای ملاحظه‌ی مستندات API بستر جاوا ۲ می‌توانید در پایگاه شرکت سان[2] به نشانی *http://java.sun.com/products/jdk/1.2/docs/api/index.html* مراجعه کنید.

[1] Software Development Kit (SDK).
[2] Sun.

فصل ۲: ساختن برنامه‌ها

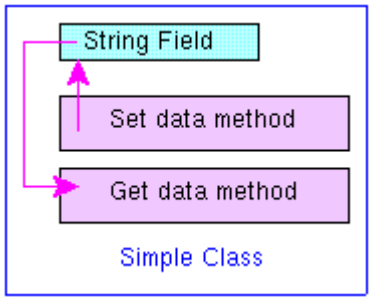
تمام برنامه‌هایی که به زبان جاوا نوشته شده‌اند (برنامه‌های جاوا) از کلاس تشکیل می‌شوند. از آنجا که همه‌ی کلاس‌ها ساختار یکسانی دارند و از عناصر مشترکی تشکیل می‌شوند، لذا تمام برنامه‌های جاوا شباهت زیادی به یکدیگر دارند.

در این فصل، ساختار و عناصر یک برنامه‌ی ساده که از یک کلاس تشکیل شده است، شرح داده می‌شود. فصل بعد همین بحث را در مورد برنامک‌ها پی می‌گیرد.

ساختار و عناصر برنامه

برنامه از کلاس تشکیل می‌شود. یک `class` از جهاتی مانند یک `RECORD` در زبان پاسکال یا یک `struct` در زبان C است، زیرا داده‌های مرتبط را در *فیلد*هایی از انواع مختلف ذخیره می‌کند. مثلاً می‌توانید در یک فیلد یک رشته‌ی متنی[۱] ذخیره کنید، در فیلد دیگر یک عدد صحیح[۲]، و در فیلد سوم یک عدد ممیز شناور[۳]. تفاوت بین یک کلاس و یک `RECORD` یا `struct` در این است که در یک کلاس *روش‌ها*یی نیز برای کار کردن بر روی این داده‌ها تعریف می‌شوند.

مثلاً در یک کلاس خیلی ساده که یک رشته‌ی متنی دارد، ممکن است یک روش برای تعیین مقدار برای رشته تعریف شود و روش دیگری برای گرفتن مقدار رشته و چاپ کردن آن روی پایانه. روش‌هایی که روی داده‌ها کار می‌کنند، روش‌های *دستیاب*[۴] نامیده می‌شوند.



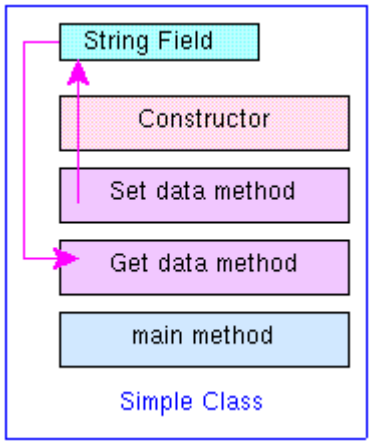
شکل ۳. یک کلاس

هر برنامه باید مشتمل بر کلاسی باشد که روشی به نام `main` داشته باشد. این کلاس نقطه‌ی ورودی برنامه است، و برای اجرای برنامه نیز باید نام این کلاس را در سطر فرمان به تفسیرگر `java` داد.

وقتی برنامه شروع می‌شود، ابتدا کد قرار گرفته در روش `main` اجرا می‌شود، و از این نقطه‌ی کنترلی است که روش‌های دستیاب کلاس کنترل‌گر[۵] برای کار کردن بر روی داده‌ها فرا خوانی می‌شوند.

---

[1] text string.
[2] integer.
[3] floating point.
[4] accessor.
[5] controller class.



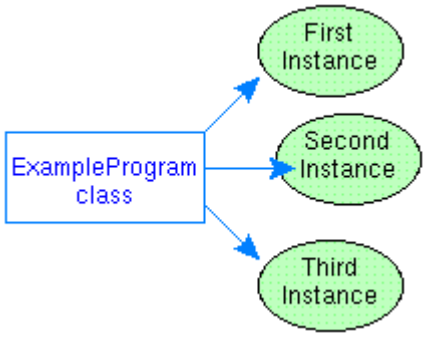
شکل ۴. روش main

در اینجا دوباره به برنامه‌ی نمونه‌ی فصل ۱ باز می‌گردیم. این کلاس هیچگونه فیلد یا روش دستیاب ندارد، ولی چون تنها کلاس برنامه است، دارای روشی به نام `main` است.

```
class ExampleProgram {
   public static void main(String[] args){
     System.out.println("I'm a Simple Program");
   }
}
```

کلیدواژه‌های `public static void` بدین معنا است که دستگاه مجازی جاوا (JVM) می‌تواند برای شروع برنامه روش `main` برنامه را فرا خواند (`public`) بدون اینکه نمونه[1]ای از کلاس بسازد (`static`) و اینکه برنامه پس از پایان یافتن داده‌ای برای تفسیرگر جاوا بر نمی‌گرداند (`void`).

نمونه‌ی کلاس نسخه‌ی قابل اجرا[2]یی از کلاس است. گرچه کلاس داده‌ها و رفتار را تعیین می‌کند، لیکن برای داشتن داده‌ها و کار کردن روی آنها باید نمونه‌ای از کلاس داشته باشید. نمودار زیر سه نمونه از کلاس `ExampleProgram` را به نام‌های `FirstInstance`، `SecondInstance` و `ThirdInstance` نشان می‌دهد.



شکل ۵. سه نمونه از یک کلاس

[1] instance.
[2] executable.

روش main از آن جهت static است که تفسیرگر JVM بتواند بدون اینکه در ابتدا نمونه‌ای از کلاس کنترل[1] بسازد، اجرای برنامه را شروع کند. نمونه‌های کلاس کنترل پس از اینکه اجرای برنامه شروع شد، در داخل روش main ایجاد می‌شوند.

روش main در مثال ساده‌ی ما نمونه‌ای از کلاس ExampleProgram ایجاد نمی‌کند، چون نیازی به آن ندارد. کلاس ExampleProgram غیر از روش main روش یا فیلد دیگری ندارد که برای دستیابی به آنها نیازی به ایجاد یک نمونه از کلاس باشد. در بستر جاوا می‌توانید بدون ایجاد نمونه‌ای از یک کلاس آن را اجرا کنید، به شرطی که روش‌های ایستا[2]ی آن هیچگونه روش غیرایستای کلاس را فرا خوانی نکنند.

کلاس ExampleProgram فقط println را فرا خوانی می‌کند که یک روش ایستا از کلاس System است. یکی از کارهای کلاس java.lang.System این است که توانایی فرستادن متن به پنجره‌ی پایانه را به ما می‌دهد. این کلاس کلاً فیلدها و روش‌های ایستا دارد.

یک برنامه می‌تواند فیلدها و روش‌های ایستای کلاس دیگری را بدون ایجاد نمونه‌ای از آن کلاس فرا خوانی کند. لذا، درست همانطور که تفسیرگر JVM می‌تواند روش main از کلاس ExampleProgram را بدون ایجاد نمونه‌ای از آن کلاس فرا خواند، کلاس ExampleProgram نیز می‌تواند روش ایستای println را از کلاس System بدون ایجاد نمونه‌ای از کلاس System فرا خوانی نماید.

با این حال، یک برنامه برای دستیابی به فیلدها و روش‌های غیرایستای یک کلاس باید نمونه‌ای از آن کلاس ایجاد کند. دستیابی به فیلدها و روش‌های ایستا و غیرایستا در قسمت بعد با مثال‌های متعددی مورد بحث قرار خواهد گرفت.

فیلدها و روش‌ها

برنامه‌ی **LessonTwoA.java** با مثال ساده‌ی قبلی از این نظر متفاوت است که رشته‌ی متنی را در یک فیلد ایستا به نام text ذخیره می‌کند. فیلد text ایستا است، بنا بر این، بدون ایجاد یک *نمونه* از کلاس LessonTwoA می‌توان به محتویات آن دستیابی حاصل کرد.

```
class LessonTwoA {
   static String text = "I'm a Simple Program";
   public static void main(String[] args){
        System.out.println(text);
   }
}
```

برنامه‌های **LessonTwoB.java** و **LessonTwoC.java** یک روش getText برای بازیابی و چاپ کردن متن به کلاس اضافه می‌کنند.

برنامه‌ی **LessonTwoB.java** برای دستیابی به فیلد غیرایستای text از روش غیرایستای getText استفاده می‌کند. روش‌ها و فیلدهای غیرایستا، روش‌ها و فیلدهای نمونه نامیده می‌شوند. این رویکرد مستلزم این است که نمونه‌ای از کلاس LessonTwoB در روش main ساخته شود. برای اینکه برنامه جالب‌تر شود، یک فیلد متنی ایستا نیز به برنامه اضافه کرده‌ایم و یک روش نمونه‌ای غیرایستا (getStaticText) نیز برای دستیابی به آن تعبیه کرده‌ایم.

توجه: فیلدها و مقادیر برگشتی روش‌ها در این مثال همه از نوع String هستند.

---

[1] control class.
[2] static.

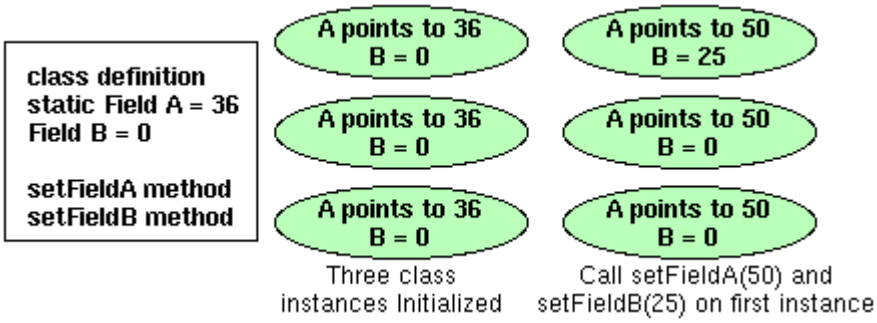
```
class LessonTwoB {

   String text = "I'm a Simple Program";
   static String text2 = "I'm static text";

   String getText(){
        return text;
   }

   String getStaticText(){
        return text2;
   }

   public static void main(String[] args){
        LessonTwoB progInstance = new LessonTwoB();
        String retrievedText = progInstance.getText();
        String retrievedStaticText =
                 progInstance.getStaticText();
        System.out.println(retrievedText);
        System.out.println(retrievedStaticText);
   }
}
```

برنامه‌ی **LessonTwoC.java** برای دستیابی به فیلد ایستای text از روش ایستای getText استفاده می‌کند. روش‌ها و فیلدهای ایستا، روش‌ها و فیلدهای کلاس خوانده می‌شوند. این رویکرد سبب می‌شود که برنامه بتواند بدون ایجاد نمونه‌ای از کلاس LessonTwoC، روش getText را فرا خواند.

```
class LessonTwoC {

   static String text = "I'm a Simple Program";

//Accessor method
   static String getText(){
        return text;
   }

   public static void main(String[] args){
        String retrievedText = getText();
        System.out.println(retrievedText);
   }
}
```

بنا بر این، روش‌های کلاس فقط می‌توانند با فیلدهای کلاس کار کنند، در حالی که روش‌های نمونه‌ای می‌توانند با فیلدهای کلاس و نمونه کار کنند.

شاید برای شما اهمیت این تفاوت مورد سؤال باشد. به طور خلاصه، از داده‌هایی که در یک فیلد کلاس قرار دارد، فقط یک نسخه وجود دارد، در حالی که هر نمونه از داده‌های مربوط به فیلدهای نمونه یک نسخه برای خود دارد.



شکل ۶. فیلدهای کلاس و فیلدهای نمونه

شکل بالا سه نمونه از یک کلاس را با یک فیلد ایستا و یک فیلد نمونه نشان می‌دهد. در زمان اجرا، یک نسخه از مقدار مربوط به فیلد ایستای `A` وجود دارد، و هر نمونه به همان نسخه اشاره می‌کند. هنگامی که نمونه‌ی اول روش `setFieldA(50)` را فرا خوانی می‌کند، مقدار فیلد از ۳۶ به ۵۰ تغییر می‌کند، و همه‌ی نمونه‌ها به مقدار جدید اشاره می‌کنند. اما با فرا خوانی `setFieldB(25)` بر روی نمونه‌ی اول، مقدار فیلد `B` فقط برای نمونه‌ی اول از ۰ به ۲۵ تغییر می‌کند، زیرا هر نمونه نسخه‌ای از این فیلد را برای خود دارد.

برای بحث کامل‌تر در این مورد به فصل "درک عضوهای کلاس و نمونه" در خودآموز جاوا به نشانی *http://java.sun.com/docs/books/tutorial/java/javaOO/classvars.html* مراجعه نمایید.

سازنده‌ها

کلاس‌ها روش خاصی به نام *سازنده*[1] دارند که هنگامی که نمونه‌ای از کلاس ساخته می‌شود، فرا خوانی می‌شود. سازنده‌ی کلاس همیشه همان نام کلاس را دارا است، و مقدار برگشتی ندارد. برنامه‌ی `LessonTwoD` برنامه‌ی `LessonTwoB` را به صورتی تغییر داده است که در آن برای آغازش[2] رشته‌ی متنی از سازنده استفاده می‌شود.

> **توجه:** اگر شما برای یک کلاس سازنده ننویسید، تدوینگر سازنده‌ای خالی اضافه می‌کند که سازنده‌ی بی‌آوند[3] کلاس والد[4] آن را فرا خوانی می‌کند. سازنده‌ی خالی را سازنده‌ی پیش‌فرض[5] می‌نامند. سازنده‌ی پیش‌فرض تمام فیلدهای و متغیرهای ناآغازیده را با مقدار صفر آغازش می‌کند.

```
class LessonTwoD {

   String text;

//Constructor
   LessonTwoD(){
     text = "I'm a Simple Program";
   }

//Accessor method
   String getText(){
        return text;
```

[1] constructor.
[2] initialization.
[3] no-arguments constructor.
[4] parent class.
[5] default constructor.

```
    }

    public static void main(String[] args){
        LessonTwoD progInst = new LessonTwoD();
        String retrievedText = progInst.getText();
        System.out.println(retrievedText);
    }
}
```

خلاصه

برنامه‌ی ساده‌ای که فقط می‌خواهد رشته‌ی متنی کوتاهی را روی پایانه بنویسد، احتمالاً همه‌ی کار را در روش `main` انجام داده و نیازی به سازنده، فیلد `text`، و روش `getText` نخواهد داشت. لیکن در این فصل ما خواستیم که با استفاده از یک برنامه‌ی بسیار ساده ساختار و عناصر یک برنامه‌ی اساسی جاوا را به شما نشان دهیم.

اطلاعات بیشتر

برای اطلاعات بیشتر به فصل "درک عضوهای کلاس و نمونه" در خودآموز جاوا به نشانی *http://java.sun.com/docs/books/tutorial/java/javaOO/classvars.html* مراجعه کنید.

فصل ۳: ساختن برنامک

برنامک‌ها هم مانند برنامه‌ها از کلاس تشکیل شده‌اند. اما برنامک‌ها روش main به عنوان نقطه‌ی شروع برنامه ندارند، بلکه در عوض روش‌های متعددی برای کنترل جنبه‌های مختلف اجرای برنامک دارند.
در این فصل، برنامه‌ای را که در فصل ۲ نوشته بودیم، به برنامک تبدیل می‌کنیم، و به تشریح ساختار و عناصر یک برنامک می‌پردازیم.

از برنامه به برنامک

کد زیر برنامک معادل برنامه‌ی LessonTwoB را از فصل ۲ نشان می‌دهد. شکل زیر اجرای برنامک را نشان می‌دهد. ساختار و عناصر کد برنامک بعد از توضیحاتی در مورد نحوه‌ی اجرای برنامک ارائه خواهد شد.

شکل ۷. برنامک

```
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Color;

public class SimpleApplet extends Applet{

  String text = "I'm a simple applet";

  public void init() {
        text = "I'm a simple applet";
        setBackground(Color.cyan);
  }
  public void start() {
        System.out.println("starting...");
  }
  public void stop() {
        System.out.println("stopping...");
  }
  public void destroy() {
        System.out.println("preparing to unload...");
  }
  public void paint(Graphics g){
```

```
        System.out.println("Paint");
        g.setColor(Color.blue);
        g.drawRect(0, 0,
                   getSize().width -1,
                   getSize().height -1);
        g.setColor(Color.red);
        g.drawString(text, 15, 25);
    }
}
```

کلاس SimpleApplet به عنوان public تعریف شده است، تا اینکه برنامه‌ای که برنامک را اجرا می‌کند (یعنی مرورگر صفحات اینترنت و یا برنامه‌ی appletviewer) بتواند به آن دسترسی پیدا کند.

اجرای برنامک

برای اینکه برنامک را در صحنه‌ی عمل ببینید، نیازمند یک پرونده‌ی HTML با برگه[1]ی Applet به صورت زیر هستید:

```
<HTML>
<BODY>
<APPLET CODE=SimpleApplet.class WIDTH=200 HEIGHT=100>
</APPLET>
</BODY>
</HTML>
```

آسان‌ترین راه برای اجرای برنامک استفاده از **appletviewer** به صورت زیر است (در اینجا simpleApplet.html نام پرونده‌ای است که حاوی کد HTML فوق است):

```
appletviewer simpleApplet.html
```

توجه: برای اینکه برنامکی را که با APIهای جاوا ۲ نوشته شده است، در یک مرورگر اجرا کنید، باید مرورگر شما مجهز به بستر جاوا ۲ باشد. اگر مرورگر مجهز به بستر جاوا ۲ ندارید، باید از **appletviewer** استفاده کنید و یا اینکه جازن جاوا[2] (نشانی *http://java.sun.com/products/plugin/index.html*) را نصب نمایید. جازن جاوا به شما اجازه می‌دهد که برنامک‌های درون صفحات اینترنتی را به جای JVM پیش‌فرض تحت ویرایش JVM ۱,۲ اجرا نمایید.

ساختار و عناصر برنامک

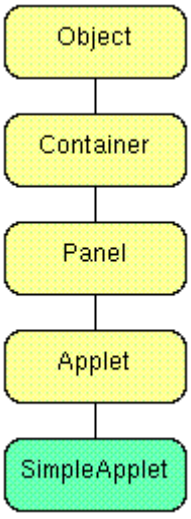
آنچه برای طراحی ظاهری و مدیریت رفتار یک برنامک لازم است، در کلاس Applet از API جاوا قرار داده شده است. این کلاس حاوی یک جزء رابط گرافیکی کاربر[3] (GUI) به نام Panel و تعدادی روش است. برای ایجاد یک برنامک، کلاس Applet را گسترش می‌دهیم (یا به عبارت دیگر، کلاسی را از آن مشتق می‌کنیم) و منظره و رفتار مورد نظر خود را پیاده‌سازی می‌نماییم.

برای ایجاد منظره‌ی ظاهری برنامک از ترسیم بر روی Panel و یا افزودن اجزای GUI دیگر مانند دکمه‌های فشاری[4]، نوارهای کرکره‌ای[5]، و یا نواحی متنی[6] به Panel استفاده می‌کنیم. رفتار برنامک با پیاده‌سازی روش‌ها تعیین می‌شود.

[1] tag.
[2] Java Plug-in.
[3] graphical user interface (GUI).
[4] push buttons.
[5] scrollbars.
[6] text areas.

گسترش دادن یک کلاس

اکثر کلاس‌ها هر چقدر ساده یا پیچیده باشند، از گسترش دادن کلاس‌های دیگر به دست می‌آیند. گسترش دادن به معنای نوشتن کلاس جدیدی است که بتواند از فیلدها و روش‌های کلاس قبلی استفاده کند. به کلاس اولیه کلاس والد[۱] و به کلاس جدید کلاس فرزند[۲] می‌گویند. به دیگر سخن، کلاس فرزند فیلدها و روش‌های کلاس والد یا زنجیره‌ی کلاس‌های والد خود را به ارث می‌برد[۳]. کلاس‌های فرزند یا همان روش‌های والد را فرا خوانی می‌کنند و یا آنها را جایگزین[۴] می‌کنند. این مفهوم را توارث منفرد[۵] می‌گویند.



شکل ۸. ساختار برنامک

کلاس `SimpleApplet` کلاس `Applet` را گسترش می‌دهد، و کلاس `Applet` کلاس `Panel` را، که آن هم به نوبه‌ی خود کلاس `Container` را گسترش می‌دهد. کلاس `Container` کلاس `Object` را گسترش می‌دهد که والد همه‌ی کلاس‌های API جاوا است.

کلاس `Applet` دارای روش‌های `init`، `start`، `stop`، `destroy`، و `paint` است که در مثال بالا دیدید. کلاس `SimpleApplet` این روش‌ها را برای انجام کارهای مورد انتظار از این برنامه جایگزین می‌کند. کلاس `Applet` قابلیت خاصی برای این روش‌ها ارائه نمی‌کند.

معذالک، کلاس `Applet` برای روش `setBackground` که در روش `init` فرا خوانی شده است، قابلیتی ارائه کرده است. فرا خواندن `setBackground` نمونه‌ای از فرا خواندن یک روش توارثی از کلاس والد به جای جایگزین کردن یک روش توارثی است.

شاید برای شما عجیب باشد که در جاوا روش‌هایی وجود دارد که پیاده‌سازی برای آن ارائه نشده است. هدف از این کار ایجاد قراردادهایی است که همه‌ی برنامه‌نویسان از آنها تبعیت کنند تا برنامه‌های جاوا از انسجام و هماهنگی برخوردار باشند. مثلاً، اگر هر کسی می‌خواست برای شروع برنامک روشی دلخواهی بنویسید، و نام متفاوتی مثلاً `begin` یا `go` به آن بدهد، دیگر برنامک بین برنامه‌ها و مرورگرهای مختلف و در بسترهای

---

[۱] parent class.
[۲] child class.
[۳] inherit.
[۴] override.
[۵] single inheritance.

گوناگون قابل اجرا نمی‌بود. مثلاً Netscape و Internet Explorer می‌دانند که برای اجرای برنامک باید به دنبال روش `init` و `start` بگردند.

رفتار

برنامک به وسیله‌ی نرم‌افزاری که آن را اجرا می‌کند، کنترل می‌شود. معمولاً این نرم‌افزار یک مرورگر[1] است، ولی ممکن است به طوری که در مثال بالا دیدید، برنامه‌ی `appletviewer` باشد. این نرم‌افزار با فرا خواندن روش‌هایی که برنامک از کلاس `Applet` به ارث می‌برد، برنامک را کنترل می‌کند.

روش init

روش `init` هنگامی که برنامک برای اولین بار ایجاد شد و به وسیله‌ی نرم‌افزار زمینه‌ای بار شد، فرا خوانده می‌شود. این روش، عملیات یکباره‌ای را که برای اجرای برنامک لازم است، مانند ایجاد رابط کاربر و یا ایجاد کردن فونت، انجام می‌دهد. در این مثال، روش `init` رشته‌ی متنی را آغازش می‌کند، و رنگ زمینه را تعیین می‌نماید.

روش start

روش `start` هنگامی که برنامک پدیدار می‌شود، مثلاً موقعی که کاربری در اینترنت وارد صفحه‌ای می‌شود که برنامکی روی آن قرار دارد. در مثال ذکر شده رشته‌ای روی پایانه چاپ می‌شود تا شما بدانید که برنامک در حال شروع شدن است. در یک برنامک پیچیده‌تر، روش `start` وظیفه‌ی انجام کارهایی را بر عهده دارد که باید در هنگام شروع برنامه اجرا شوند، مانند شروع کردن یک متحرک‌سازی[2] یا نواختن یک آهنگ.

پس از اجرای روش `start`، ریسه‌ی رویدادها[3] روش `paint` را فرا خوانی می‌کند تا `Panel` برنامک را ترسیم نماید. ریسه به یک جریان کنترلی منفرد در درون برنامک می‌گویند. هر برنامک می‌تواند در ریسه‌های متعدد اجرا شود. روش‌های ترسیمی `Applet` همیشه از درون یک ریسه‌ی اختصاصی ترسیم کننده و رویدادپرداز فرا خوانده می‌شوند.

روش‌های stop و destroy

روش `stop` هنگامی که برنامک دیگر روی صفحه‌ی نمایش نیست، مثلاً زمانی که کاربر به صفحه‌ی دیگری می‌رود، برنامک را متوقف می‌کند. در مثال ما، این روش رشته‌ای را روی پایانه می‌نویسد تا به شما بگوید که برنامک در حال متوقف شدن است. در یک برنامک پیچیده‌تر، این روش باید کارهایی را از قبیل متوقف کردن متحرک‌سازی یا آهنگ انجام دهد.

روش `destroy` هنگامی فرخوانده می‌شود که برنامه‌ی مرورگر خاتمه می‌پذیرد. برنامک شما باید این روش را برای انجام پاکسازی‌های نهایی مانند متوقف کردن ریسه‌های زنده پیاده‌سازی کند.

---

[1] browser.
[2] animation.
[3] the event thread.

منظره‌ی ظاهری

عضو Panel در کلاس Applet یک روش paint از کلاس والد خود Container به ارث می‌برد. برای اینکه چیزی روی Panel برنامک بکشید، باید روش paint را پیاده‌سازی کنید.
شیئ Graphics که به عنوان پارامتر به روش paint داده می‌شود، یک *قرینه‌ی گرافیکی* برای کشیدن روی Panel تعیین می‌کند. این شیئ روش‌هایی برای عملیات گرافیکی مانند تعیین رنگ‌های ترسیم، و کشیدن گرافیک‌ها، تصویرها، و متن دارد.
روش paint از SimpleApplet رشته‌ی "I'm a simple applet" را به رنگ قرمز در درون یک مستطیل آبی ترسیم می‌کند.

```
   public void paint(Graphics g){
        System.out.println("Paint");
//Set drawing color to blue
        g.setColor(Color.blue);
//Specify the x, y, width and height for a rectangle
        g.drawRect(0, 0,
                   getSize().width -1,
                   getSize().height -1);
//Set drawing color to red
        g.setColor(Color.red);
//Draw the text string at the (15, 25) x-y location
        g.drawString(text, 15, 25);
   }
```

بسته‌ها

در بالای کد این برنامک سه دستورالعمل import دیده می‌شود. برنامه‌های کوچک و بزرگ و برنامک‌ها از دستورالعمل import برای دسترسی به کلاس‌های آماده‌ی API جاوا که در دورن *بسته‌ها*[1] قرار گرفته‌اند، استفاده می‌کنند. این مطلب هم در مورد کلاس‌های API جاوا که شما در حین فروگذاری بستر جاوا دریافت کرده‌اید صحت دارد، هم در مورد کلاس‌هایی که از طرف ثالث می‌گیرید، و هم در مورد کلاس‌هایی که خودتان می‌نویسید و آنها را در پوشه‌ای جدا از برنامه ذخیره می‌کنید. در زمان تدوین، برنامه از دستورالعمل import برای تعیین محل اشاره به کلاس‌های تدوین شده‌ی جاوا که در بسته‌هایی روی ماشین محلی و یا سامانه‌ی شبکه‌ای ذخیره شده‌اند، استفاده می‌کند. ممکن است نام یک کلاس تدوین شده در یک بسته با نام کلاس دیگر ید بسته‌ی دیگر یکسان باشد. نام بسته این دو کلاس را از یکدیگر افتراق می‌دهد.
مثال‌های فصل‌های ۱ و ۲ برای فرا خواندن روش System.out.println از کلاس‌های API جاوا نیازی به استفاده از دستورالعمل import نداشتند، زیرا کلاس System در بسته‌ی java.lang قرار دارد که به طور پیش‌فرض فرا خوانده می‌شود.

اطلاعات بیشتر

برای اطلاعات بیشتر در باره‌ی نوشتن برنامک‌ها به فصل "نوشتن برنامک‌ها" در خودآموز جاوا به نشانی *http://java.sun.com/docs/books/tutorial/applet/* مراجعه کنید.

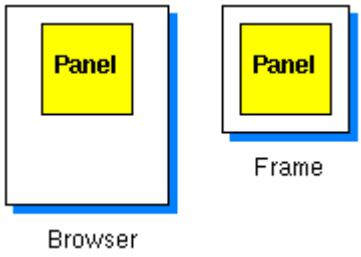
[1] packages.

فصل ۴: ساختن رابط کاربر

در فصل قبل دیدید که کلاس `Applet` یک جزء `Panel` دارد که با استفاده از آن می‌توانید رابط کاربر را طراحی کنید. در این درس، برنامه‌ی اساسی فصل‌های ۱ و ۲ را گسترش می‌دهیم و با استفاده از API پروژه‌ی سوینگ[1] از کلاس‌های بنیادی جاوا[2] (JFC) یک رابط کاربر برای آن می‌سازیم، که به پردازش رویدادهای مربوط به کاربر می‌پردازد.

API سوینگ

بر خلاف برنامکی که در فصل ۳ ساختیم و در آن رابط کاربر متصل به یک پانل[3] بود که در درون یک مرورگر سطح بالا واقع شده بود، برنامه‌ی سوینگی که در این فصل خواهیم نوشت، رابط کاربر خود را به شیئ پانلی متصل می‌کند که در درون یک فریم[4] سطح بالا قرار گرفته است. شیئ فریم پنجره‌ی سطح بالایی است که دارای عنوان و روش‌هایی برای مدیریت منظره و رفتار فریم است.



شکل ۹. پانل و فریم

کد زیر برنامه‌ی ساده‌ی مورد نظر ما را تشکیل می‌دهد. پنجره‌ی طرف چپ هنگام شروع برنامه ظاهر می‌شود، و پنجره‌ی طرف راست هنگامی پدیدار می‌شود که دکمه را بزنید. اگر دوباره دکمه را بزنید، به پنجره‌ی اول باز می‌گردید.

شکل ۱۰. برنامه‌ی سوینگ در هنگام شروع (چپ) و پس از زدن دکمه (راست)

[1] Swing.
[2] Java™ Foundation Classes (JFC).
[3] panel.
[4] frame.

دستورالعمل‌های import

متن پرونده‌ی *SwingUI.java* را می‌توانید در بخش متن برنامه‌ها ببینید (صفحه‌ی ۸۹). در بالای این متن، چهار دستورالعمل import جلب نظر می‌کند. این دستورالعمل‌ها دقیقاً مشخص می‌کنند که برنامه از چه کلاس‌هایی از API جاوا استفاده می‌کند. می‌توانستیم سه تا از این دستورالعمل‌ها را با یک دستورالعمل import java.awt.* جایگزین کنید تا تمام بسته‌ی awt را وارد کند، اما این کار نسبت به زمانی که دقیقاً کلاس‌های مورد نظر را مشخص می‌کنیم، سربار زمان تدوین را افزایش می‌دهد.

```
import java.awt.Color;
import java.awt.BorderLayout;
import java.awt.event.*;
import javax.swing.*;
```

اعلام کلاس

بعد از دستورالعمل فوق، اعلام کلاس واقع شده است و نشان می‌دهد که فریم سطح بالای رابط کاربر برنامه، یک Jframe است که رابط ActionListener را پیاده‌سازی می‌کند.

```
class SwingUI extends JFrame
        implements ActionListener{
```

کلاس Jframe، کلاس Frame را گسترش می‌دهد که جزئی از API جعبه‌ابزار پنجره‌ی انتزاعی[1] (AWT) است. پروژه‌ی سوینگ[2] AWT را گسترش می‌دهد و مجموعه‌ی کاملی از اجزا[3] و خدمات[4] رابط گرافیکی کاربر (GUI)، قابلیت منظره‌ی کلی جازدنی[5]، و پشتیبانی از فناوری کمکی[6] را به آن می‌افزاید. برای آشنایی بیشتر با سوینگ می‌توانید به "مرکز روابط سوینگ" به نشانی *http://java.sun.com/products/jfc/tsc/index.html* و به بخش اول از راهنمای "مبانی سوینگ" به نشانی *http://java.sun.com/developer/onlineTraining/GUI/Swing1/index.html* مراجعه کنید.

API جاوا حاوی کلاس‌ها و رابط‌هایی است که می‌توانید از آنها استفاده کنید. یک رابط، مجموعه‌ای از روش‌ها را تعریف می‌کند ولی آنها را پیاده‌سازی نمی‌کند. د راعلام کلاس SwingUI مشخص شده است که این کلاس رابط ActionListener را پیاده‌سازی می‌کند. بر این اساس، کلاس SwingUI باید تمام روش‌هایی را که در رابط ActionListener تعریف شده‌اند، پیاده‌سازی کند. خوشبختانه یک روش بیشتر وجود ندارد و آن هم روش actionPerformed است که در زیر در باره‌ی آن بحث می‌کنیم.

متغیرهای نمونه‌ای

در چند سطر بعد اجزایی از سوینگ که کلاس SwingUI از آنها استفاده می‌کند، اعلام شده‌اند. اینها متغیرهای نمونه‌ای هستند که به وسیله‌ی هر روشی در کلاس نمونه‌سازی[7] شده قابل استفاده هستند. در این مثال، این اجزا در سازنده‌ی SwingUI ساخته می‌شوند و در پیاده‌سازی روش actionPerformed

[1] Abstract Window Toolkit (AWT).
[2] Swing Project.
[3] components.
[4] services.
[5] pluggable look and feel.
[6] assistive technology.
[7] instantiate.

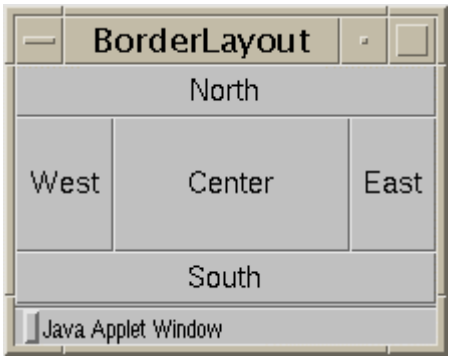
مورد استفاده قرار می‌گیرند. متغیر نمونه‌ای که به صورت `private boolean` تعریف شده است فقط برای کلاس `SwingUI` قابل دسترسی است و روش `actionPerformed` برای فهمیدن اینکه دکمه فشار داده شده است یا نه، از آن استفاده می‌کند.

```
JLabel text, clicked;
JButton button, clickButton;
JPanel panel;
private boolean _clickMeMode = true;
```

سازنده

سازنده که در پایین نشان داده شده است، اجزای رابط کاربر را ایجاد می‌کند، این اجزا را به شیئ `JPanel` اضافه می‌کند، پانل را به فریم اضافه می‌کند، و رویدادشنو[1] جزء `JButton` را اضافه می‌کند. شیئ `JFrame` هنگام شروع برنامه در روش `main` ساخته می‌شود.

```
SwingUI(){
  text = new JLabel("I'm a Simple Program");
  clicked = new JLabel("Button Clicked");

  button = new JButton("Click Me");
//Add button as an event listener
  button.addActionListener(this);

  clickButton = new JButton("Click Again");
//Add button as an event listener
  clickButton.addActionListener(this);

//Create panel
  panel = new JPanel();
//Specify layout manager and background color
  panel.setLayout(new BorderLayout(1,1));
  panel.setBackground(Color.white);
//Add label and button to panel
  getContentPane().add(panel);
  panel.add(BorderLayout.CENTER, text);
  panel.add(BorderLayout.SOUTH, button);
}
```



شکل ۱۱. چینش `BorderLayout`

هنگامی که شیئ `JPanel` ایجاد شد، مدیر چینش[1] و رنگ زمینه تعیین می‌شوند. مدیر چینش مورد استفاده چگونگی چینش اجزای رابط کاربر را در ناحیه‌ی نمایش تعیین می‌کند.

[1] event listener.

در این برنامه از مدیر چینش BorderLayout استفاده شده است، که اجزای رابط کاربر را در پنج ناحیه قرار می‌دهد. برای اضافه کردن یک جزء، ناحیه‌ی مورد نظر را مشخص کنید (شمال، جنوب، شرق، غرب، و یا مرکز).

```
//Create panel
     panel = new JPanel();
//Specify layout manager and background color
     panel.setLayout(new BorderLayout(1,1));
     panel.setBackground(Color.white);
//Add label and button to panel
     getContentPane().add(panel);
     panel.add(BorderLayout.CENTER, text);
     panel.add(BorderLayout.SOUTH, button);
   }
```

برای اطلاع از سایر انواع مدیر چینش و چگونگی استفاده از آنها به مقاله‌ی "بررسی مدیرهای چینش AWT" به نشانی *http://java.sun.com/developer/technicalArticles/GUI/AWTLayoutMgr/index.html* مراجعه کنید.

برای اضافه کردن Panel به Jframe، تابع getContentPane فرا خوانده شده است. اجزا مستقیماً به یک Jframe اضافه نمی‌شوند، بلکه به صفحه‌ی محتویات (content pane) آن اضافه می‌شوند. از آنجا که مدیر چینش مسئول تعیین چینش اجزا است، لذا باید در جایی قرار داده شود که اجزا قرار دارند. صفحه‌ی محتویات امکان آن را فراهم می‌کند که انواع مختلف اجزا در پروژه‌ی سوینگ با هم کار کنند.

شنیدن عمل‌ها

علاوه بر پیاده‌سازی رابط ActionListener، لازم است که رویدادشنو را به اجزای JButton اضافه کنید. رویدادشنو شیئ SwingUI است، زیرا رابط ActionListener را پیاده‌سازی کرده است. در این مثال، هنگامی که کاربر دکمه را فشار می‌دهد، سرویس‌های مربوطه‌ی بستر جاوا این عمل (رویداد) را روش actionPerformed تحویل می‌دهند. شما در برنامه‌ی خود باید روش actionPerformed را به گونه‌ای پیاده‌سازی کنید که بر اساس اینکه چه دکمه‌ای فشار داده شده است، عمل مناسب را انجام دهد.

هر کلاس جزء روشی برای اضافه کردن رویدادشنوها به آن دارد. در این برنامه، جزء JButton یک روش addActionListener دارد. پارامتری که به این تابع داده می‌شود، this است، به عبارت دیگر، عمل‌شنو SwingUI به دکمه اضافه می‌شود، تا عمل‌های تولید شده به وسیله‌ی دکمه به روش actionPerformed در شیئ SwingUI داده شود.

```
     button = new JButton("Click Me");
//Add button as an event listener
     button.addActionListener(this);
```

رویدادپردازی

به روش actionPerformed شیئ رویدادی داده می‌شود که نشان دهنده‌ی رویداد عملی است که اتفاق افتاده است. این روش با استفاده از یک دستورالعمل if نگاه می‌کند ببیند کدام جزء این رویداد را ایجاد کرده است، و بر اساس نتیجه‌ی این بررسی، عمل مناسب را انجام می‌دهد:

---

[1] layout manager.

```
  public void actionPerformed(ActionEvent event){
    Object source = event.getSource();
       if (_clickMeMode) {
         text.setText("Button Clicked");
         button.setText("Click Again");
         _clickMeMode = false;
       } else {
         text.setText("I'm a Simple Program");
         button.setText("Click Me");
         _clickMeMode = true;
       }
  }
```

برای کسب اطلاعات بیشتر در باره‌ی رویدادپردازی در اجزای مختلف به "خودآموز جاوا" قسمت "رویدادپردازی" به نشانی *http://java.sun.com/docs/books/tutorial/ui/swingOverview/event.html* مراجعه کنید.

روش main

روش main فریم اصلی را ایجاد می‌کند، عنوان آن را تعیین می‌کند، و امکان آن را فراهم می‌کند که کاربر نهایی با استفاده از فهرست[1] پنجره بتواند پنجره را ببندد.

```
public static void main(String[] args){
//Create top-level frame
  SwingUI frame = new SwingUI();
  frame.setTitle("Example");
//This code lets you close the window
  WindowListener l = new WindowAdapter() {
     public void windowClosing(WindowEvent e) {
       System.exit(0);
     }
  };
    frame.addWindowListener(l);
//This code lets you see the frame
  frame.pack();
  frame.setVisible(true);
 }
}
```

کد مربوط به بستن پنجره راه آسانی را برای افزودن قابلیت رویدادپردازی به یک برنامه نشان می‌دهد. در صورتی که رابط رویدادشنوی مورد نظر شما قابلیت‌های بیشتری نسبت به آنچه شما واقعاً نیاز دارید، در بر دارد، به جای آن از یک کلاس سازگار کننده[2] استفاده کنید. در API جاوا برا تمام رابط‌های شنونده که بیش از یک روش دارند، کلاس‌های سازگار کننده وجود دارد. بدین ترتیب، می‌توانید به جای رابط شنونده، از کلاس سازگار کننده استفاده کنید و فقط روشی را که می‌خواهید، پیاده‌سازی نمایید. در این مثال، رابط WindowListener ۷ روش دارد و این برنامه فقط به روش windowClosing نیاز دارد، بدین خاطر است که از کلاس WindowAdapter استفاده شده است.

برنامه کلاس WindowAdapter راگسترش داده و روش windowClosing را پیاده‌سازی می‌نماید. کلیدواژه‌ی new یک نمونه‌ی بی‌نام از کلاس درونی گسترش داده شده ایجاد می‌کند. این کلاس بی‌نام است،

[1] menu.
[2] adapter class.

زیرا نامی به آن داده نشده است، و بدون اجرای مجدد این کد نمی‌توان نمونه‌ی دیگری از آن ایجاد کرد. از طرف دیگر، این کلاس یک کلاس درونی است چون تعریف آن در درون کلاس SwingUI واقع شده است.

با این روش فقط چند سطر برنامه‌نویسی لازم است، در حالی که اگر می‌خواستیم رابط WindowListener را پیاده‌سازی کنیم، باید ۶ روش خالی را پیاده‌سازی می‌کردیم. یادتان نرود که شیئ WindowAdapter را به شیئ فریم اضافه کنید تا رویدادهای پنجره به شیئ فریم منتقل شود.

```
  WindowListener l = new WindowAdapter() {
//The instantiation of object l is extended to
//include this code:
    public void windowClosing(WindowEvent e){
       System.exit(0);
    }
  };
  frame.addWindowListener(l);
```

نگاهی مجدد به برنامک‌ها

با استفاده از آنچه در فصل ۳ و فصل حاضر آموختید، مثال این درس (*SwingUI.java*) را به یک برنامک تبدیل کنید. اول سعی کنید این کار را خودتان انجام دهید و سپس به حل آن (*ApptoAppl.java*) مراجعه کنید.

فصل ۵: نوشتن سرولت

سرولت برنامه‌ای است که بر قابلیت‌های یک سرور می‌افزاید. شایع‌ترین استفاده از سرولت، ددان توانایی ایجاد صفحه‌های اینترنتی پویا به سرور است. سرورهای شبکه از سندهایی که به زبان نشانه‌گذاری ابرمتن[۱] (HTML) نوشته شده استفاده می‌کنند، و به تقاضاهای کاربر بر اساس پروتکل انتقال ابرمتن (HTTP) پاسخ می‌دهند. HTTP پروتکل انتقال پرونده‌های ابرمتن در اینترنت است. سندهای HTML حاوی متن نشانه‌گذاری شده هستند که باید به وسیله‌ی یک مرورگر[۲] HTML مانند نتاسکیپ[۳] تفسیر شود.

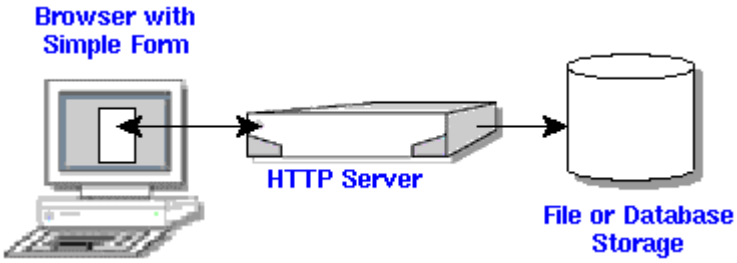
نوشتن سرولت راحت است. تنها چیزی که لازم دارید، داشتن نرم‌افزار بستر جاوا ۲ و کیت برنامه‌نویسی اینترنت جاواسرور[۴] (JWSDK) است. JWSDK را می‌توانید به رایگان از نشانی *http://java.sun.com/products/servlet/index.html* فروگذاری کنید.

در این فصل فرا خواهید گرفت که چگونه فرم خیلی ساده‌ای ایجاد کنید که یک سرولت اساسی را برای پردازش داده‌هایی که کاربر در فرم وارد کرده است، فرا خوانی می‌کند.

در باره‌ی این مثال

مرورگر داده‌های کاربر را از طریق یک فرم HTML دریافت می‌کند. فرم ساده‌ای که در این فصل مورد استفاده قرار گرفته است، یک فیلد ورودی متن دارد که کار می‌تواند در آن متنی را بنویسد و یک دکمه‌ی `Submit` (تحویل) دارد. هنگامی که کاربر نهایی دکمه‌ی تحویل را می‌زند، سرولت ساده فرا خوانی می‌شود تا متن ورودی کاربر را پردازش نماید.

در این مثال، سرولت ساده یک صفحه‌ی HTML بر می‌گرداند که حاوی متن وار شده به وسیله‌ی کاربر است.

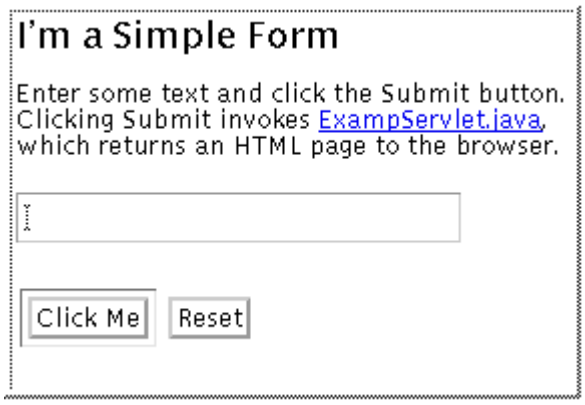


شکل ۱۲. سرولت

فرم HTML در یک پرونده‌ی HTML به نام *simpleHTML.html* قرار گرفته است. نمودار زیر تصویر این صفحه را پس از باز کردن در یک مرورگر نشان می‌دهد.

---

[۱] HyperText Markup Language (HTML).
[۲] browser.
[۳] Netscape.
[۴] JavaServer™ Web Development Kit (JWSDK).



شکل ۱۳. فرم

پرونده‌ی HTML و فرم آن تا حد زیادی شبیه برنامه و برنامکی است که در فصل ۴ به عنوان نمونه ارائه شد. بنا بر این، می‌توانید کد این برنامه‌ها را با هم مقایسه کنید تا بفهمید که سرولت‌ها، برنامک‌ها، و برنامه‌ها چگونه با ورودی کاربر نهایی برخورد می‌کنند. هنگامی که کاربر دکمه‌ی `Click Me` از می‌زند، سرولت متن ورودی را می‌گیرد و یک صفحه‌ی HTML را که حاوی آن متن است، بر می‌گرداند. صفحه‌ی HTMLای که به وسیله‌ی سرولت *ExampServlet.java* به مرورگر بر گردانده می‌شود، در زیر نشان داده شده است. اینک به بحث در باره‌ی قسمتی از کد سرولت می‌پردازیم که ورودی کاربر را می‌گیرد و صفحه‌ی HTML را ایجاد می‌کند.

شکل ۱۴. پاسخ سرولت

توجه: برای اجرای این مثال، باید سرولت و پرونده‌های HTML را در شاخه‌ی مناسب که به وسیله‌ی سرور شبکه تعیین شده است، قرار دهید. مثلاً در Java WebServer 1.1.1 سرولت در شاخه‌ی `~/JavaWebServer1.1.1/servlets` و پرونده‌ی HTML در شاخه‌ی `~/JavaWebServer1.1.1/public_html` قرار داده می‌شود.

ساختار سرولت

ExampServlet.java یک صفحه‌ی HTML ایجاد می‌کند که به کاربر نهایی باز گردانده می‌شود. بر این اساس، کد سرولت از اجزای سوینگ یا AWT استفاده نمی‌کند و حاوی کد رویدادپردازی نیز نیست. برای این سرولت بسیار ساده، لازم است که بسته‌های زیر را وارد کنید:

- `java.io` برای ورودی و خروجی سیستم. کلاس `HttpServlet` از کلاس `IOException` موجود در این بسته برای نشان دادن اینکه استثنایی بروز کرده است، استفاده می‌کند.

- javax.servlet، که حاوی کلاس‌های سرولت عمومی (مستقل از پروتکل) است. HttpServlet از کلاس ServletException موجود در این بسته برای نشان دادن مشکلات مربوط به سرولت استفاده می‌کند.
- javax.servlet.http، که حاوی کلاس‌های سرولت HTTP است. کلاس HttpServlet در این بسته قرار گرفته است.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ExampServlet extends HttpServlet {
  public void doPost(HttpServletRequest request,
   HttpServletResponse response)
        throws ServletException, IOException
  {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    out.println("<title>Example</title>" +
       "<body bgcolor=FFFFFF>");

    out.println("<h2>Button Clicked</h2>");

    String DATA = request.getParameter("DATA");

    if(DATA != null){
      out.println(DATA);
    } else {
      out.println("No text entered.");
    }

    out.println("<P>Return to
  <A HREF="../simpleHTML.html">Form</A>");
    out.close();
  }
}
```

اعلام کلاس و روش‌ها

تمام کلاس‌های سرولت کلاس انتزاعی HttpServlet را گسترش می‌دهند. HttpServlet با فراهم کردن یک چارچوب کاری برای کار کردن با HTTP، نوشتن سرولت‌های HTTP را آسان می‌کند. از آنجا که HttpServlet یک کلاس abstract است، لذا کلاس سرولت شما باید آن را گسترش دهد و لااقل یکی از روش‌هایش را پیاده‌سازی کند. کلاس abstract کلاسی است که حاوی روش‌های پیاده‌سازی نشده است، و نمی‌توان خود آن را نمونه‌سازی کرد.

```
public class ExampServlet extends HttpServlet {
  public void doPost(HttpServletRequest request,
         HttpServletResponse response)
         throws ServletException, IOException
  {
```

کلاس ExampServlet از آن رو به عنوان public اعلام شده است که سرور شبکه که سرور را اجرا می‌کند، و در شاخه‌ای غیر از شاخه‌ی سرولت واقع شده است، بتواند به آن دسترسی پیدا کند.

کلاس ExampServlet یک روش doPost تعریف می‌کند که از نظر نام، نوع برگشتی، و لیست پارامترها مانند روش doPost در کلاس HttpServlet است. بدین ترتیب، کلاس ExampServlet روش doPost در کلاس HttpServlet را جایگزین[1] و پیاده‌سازی می‌کند.

روش doPost عمل POST از پروتکل HTTP را انجام می‌دهد. همین عمل در فرم HTML برنامه‌ی مورد نظر ما مورد استفاده قرار گرفته است. عمل دیگر در این ارتباط عمل HTTP GET است که در صورت استفاده از آن باید روش doGet را مورد استفاده قرار دهید.

به طور خلاصه، کار POST فرستادن هر مقدار از داده‌ها از طریق اتصال شبکه بدون تغییر دادن URL[2] است، و GET به معنای گرفتن مقدار محدودی از اطلاعات است که به انتهای URL اضافه می‌شود. تقاضاهای POST را نمی‌توان نشانه‌گذاری[3] کرد یا ایمیل نمود، و این تقاضاها URL پاسخ را تغییر نمی‌دهند. تقاضاهای GET قابل نشانه‌گذاری و ایمیل کردن هستند و اطلاعاتی به URL پاسخ اضافه می‌کنند.

لیست پارامترهای روش doPost شامل یک شیئ request و یک شیئ response است. مرورگر تقاضایی برای سرولت می‌فرستد، و سرولت پاسخ را به مرورگر باز می‌گرداند.

پیاده‌سازی روش doPost بدین صورت است که این روش ابتدا به شیئ request توجه می‌کند ببیند چه کسی تقاضا را فرستاده است، داده‌های تقاضا به چه صورت است، و سربرگ[4]های HTTP فرستاده شده چیست. سپس با استفاده از شیئ response یک صفحه‌ی HTML در پاسخ به تقاضای مرورگر ایجاد می‌کند. روش doPost در صورتی که مشکلی در حین ورودی و خروجی پیش آید، یک استثنای IOException صادر می‌کند، و اگر تقاضا قابل رسیدگی نباشد، یک استثنای ServletException صادر می‌نماید. به این استثناها در کلاس HttpServlet رسیدگی می‌شود.

پیاده‌سازی روش

در اولین قسمت doPost با استفاده از شیئ response یک صفحه‌ی HTML ایجاد می‌شود. سپس نوع محتوی پاسخ را از نوع text/html تعیین می‌نماید، و یک شیئ PrintWriter برای صادر کردن خروجی متنی فرمت شده از پاسخ می‌گیرد.

```
response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    out.println("<title>Example</title>" +
       "<body bgcolor=#FFFFFF>");

    out.println("<h2>Button Clicked</h2>");
```

سطر بعدی با شیئ request داده‌ها را از فیلد متنی فرم می‌گیرد و آن را در متغیر DATA ذخیره می‌کند. روش getparameter مقدار پارامتر داده شده را بر می‌گرداند. اگر پارامتر وجود نداشته باشد، مقدار null بر می‌گرداند، و اگر مقداری برای آن تعیین نشده باشد، یک رشته‌ی خالی بر می‌گرداند.

```
    String DATA = request.getParameter("DATA");
```

قسمت بعدی روش doPost داده‌های لازم را از پارمتر DATA استخراج می‌کند و آن را به شیئ response می‌دهد که در صفحه‌ی پاسخ HTML وارد نماید.

```
    if(DATA != null){
```

[1] override.
[2] Uniform Resource Locators (URL).
[3] bookmark.
[4] header.

```
      out.println(DATA);
    } else {
      out.println("No text entered.");
    }
```

قسمت آخر روش doPost پیوندی ایجاد می‌کند که کاربر نهایی با استفاده از آن می‌تواند به صفحه‌ی قبلی که فرم اصلی در آن قرار داشت برگردد. سپس پاسخ را می‌بندد.

```
    out.println("<P>Return to
        <A HREF="../simpleHTML.html">Form</A>");
    out.close();
  }
```

توجه: برای فرا گرفتن سایر روش‌های موجود در کلاس‌های HttpServlet، HttpServletRequest، و HttpServletResponse به "خودآموز جاوا" درس "سرولت‌ها" به نشانی *http://java.sun.com/docs/books/tutorial/servlets/index.html* مراجعه کنید.

اطلاعات بیشتر

اطلاعات بیشتر در مورد سرولت‌ها را می‌توانید در درس "سرولت‌ها" از "خودآموز جاوا" به نشانی *http://java.sun.com/docs/books/tutorial/servlets/index.html* بیابید.

فصل ۶: دستیابی به پرونده و اجازه‌ها

تا اینجا یاد گرفتید که چگونه رشته‌ی متنی کوتاهی را که از طریق صفحه کلید وارد شده است، بگیرید و آن را در رابط گرافیکی کاربر (GUI) مورد استفاده قرار دهید. با این حال، برنامه‌ها داده‌ها را در پرونده‌ها و پایگاه‌های داده‌ای نیز ذخیره، پردازش، و بازیابی می‌کنند.

در این فصل، برنامه‌های فصل‌های قبل را به گونه‌ای تغییر می‌دهیم که با استفاده از API موجود در بسته‌ی `java.io` کارهای اساسی دستیابی به پرونده را انجام دهند. همچنین، چگونگی دادن اجازه‌ی دسترسی به پرونده‌های خاص به برنامک‌ها و محدود کردن دسترسی یک برنامه به پرونده‌های خاص را نیز بیان خواهیم کرد.

دستیابی به پرونده‌ها به وسیله‌ی برنامه‌ها

نرم‌افزار بستر جاوا ۲ کلاس‌های مختلفی برای خواندن داده‌ها به صورت نویسه‌ای یا بایتی به وسیله‌ی برنامه و نوشتن داده‌های نویسه‌ای یا بایتی در یک پرونده‌ی خارجی، دستگاه ذخیره‌سازی، و یا برنامه ارائه می‌کند. منبع یا مقصد داده‌ها ممکن است روی دستگاه رایانه‌ای باشد که برنامه روی آن اجرا می‌شود و یا اینکه هر جای دیگری از شبکه قرار داشته باشد.

در این بخش چگونگی خواندن و نوشتن داده‌ها در یک پرونده در دستگاه رایانه‌ی محلی را خواهید آموخت. برای اطلاعات بیشتر در باره‌ی چگونگی انتقال داده‌ها بین برنامه‌ها یا بین برنامه و حافظه، و انجام عملیاتی مانند بافر کردن یا کد کردن نویسه‌ها در موقع خواندن و نوشتن به درس "خواندن و نوشتن" از "خودآموز جاوا" به نشانی *http://java.sun.com/docs/books/tutorial/essential/io/index.html* مراجعه نمایید.

- **خواندن:** برنامه یک *جویبار*[1] ورودی ورودی روی پرونده باز می‌کنند و داده‌ها را به طور متوالی (به همان ترتیبی که نوشته شده است)، می‌خواند.
- **نوشتن:** برنامه یک جویبار خروجی روی پرونده باز می‌کند و داده‌ها را به طور متوالی می‌نویسد.

مثال اول برنامه‌ی *SwingUI.java* از فصل ۴ را به گونه‌ای تغییر می‌دهد که ورودی کاربر را از طریق یک فیلد متنی[2] دریافت کند. پنجره‌ی طرف چپ زمانی که برنامه‌ی *FileIO* را آغاز می‌کنید، ظاهر می‌شود، و پنجره‌ی طرف راست هنگامی که دکمه را می‌زنید. وقتی دکمه را می‌زنید، متن وارد شده در فیلد متنی در یک پرونده ذخیره می‌شود. سپس، پورنده‌ی دیگری باز و خوانده می‌شود، و متن آن در پنجره‌ای که در طرف راست نشان داده شده است، نمایش داده می‌شود. وقتی دوباره دکمه را بزنید، به پنجره‌ی اول باز می‌گردید و فیلد متنی خالی آماده است که دوباره متنی در آن وارد کنید.

[1] stream.
[2] text field.

هنگام شروع برنامه

پس از زدن دکمه

شکل ۱۵. ورودی و خروجی پرونده

به طوری که در زیر خواهیم گفت، تبدیل *SwingUI.java* از فصل ۴ به برنامه‌ی *FileIO.java* در این فصل عمدتاً در سازنده و روش `actionPerformed` صورت می‌پذیرد.

تغییرات سازنده و متغیر نمونه‌ای

یک متغیر نمونه‌ای از نوع `JtextField` به کلاس اضافه می‌شود تا سازنده بتواند آن شیئ را نمونه‌سازی[1] کند و روش `actionPerformed` بتواند به متنی که کاربر نهایی در آن تایپ می‌کند، دسترسی پیدا کند. سازنده `JTextField` را با مقدار 20 نمونه‌سازی می‌کند. این مقدار به بستر جاوا می‌گوید که برای محاسبه‌ی عرض ترجیحی فیلد، تعداد ستون‌های آن را ۲۰ بگیرد. اگر این مقدار کمتر باشد، فیلد باریک‌تر می‌شود، و اگر بزرگ‌تر باشد، پهن‌تر می‌گردد.
نوشته[2]ی `text` به قسمت `NORTH` از `BorderLayout` افزوده می‌شود، و لذا `JTextField` را می‌توان به قسمت `CENTER` اضافه کرد.

> **توجه:** برای آگاهی بیشتر در باره‌ی تعیین اندازه‌ی اجزا به خود آموز جاوا (*http://java.sun.com/docs/books/tutorial*) قسمت‌های "حل مسایل شایع چینش" (به نشانی *http://java.sun.com/docs/books/tutorial/ui/swingLayout/problems.html*) و "مدیریت چینش" (به نشانی *http://java.sun.com/docs/books/tutorial/ui/swingOverview/layout.html*) مراجعه کنید.

```
//Instance variable for text field
JTextField textField;

FileIO(){
  text = new JLabel("Text to save to file:");
  clicked = new
   JLabel("Text retrieved from file:");

  button = new JButton("Click Me");
  button.addActionListener(this);

  clickButton = new JButton("Click Again");
  clickButton.addActionListener(this);

//Text field instantiation
  textField = new JTextField(20);
```

[1] instantiate.
[2] label.

```
  panel = new JPanel();
  panel.setLayout(new BorderLayout());
  panel.setBackground(Color.white);
  getContentPane().add(panel);

//Adjustments to layout to add text field
  panel.add("North", text);
  panel.add("Center", textField);
  panel.add("South", button);
}
```

تغییرات روش‌ها

روش actionPerformed از کلاس‌های FileInputStream و FileOutputStream برای خواندن و نوشتن داده‌های یک پرونده استفاده می‌کند. این کلاس‌ها داده‌های را به صورت جویبارهای بایتی[1]، و نه به صورت جویبارهای نویسه‌ای[2] که در مثال برنامک دیدیم، پردازش می‌کنند. پس از متن کد زیر، توضیح مفصل‌تری در باره‌ی تغییرات پیاده‌سازی روش ارائه می‌نماییم.

```
public void actionPerformed(
     ActionEvent event){
  Object source = event.getSource();
  if(source == button){
//Variable to display text read from file
    String s = null;
    if(_clickMeMode){
      try{
//Code to write to file
        String text = textField.getText();
        byte b[] = text.getBytes();

        String outputFileName =
     System.getProperty("user.home",
     File.separatorChar + "home" +
     File.separatorChar + "monicap") +
     File.separatorChar + "text.txt";
          File outputFile = new File(outputFileName);
          FileOutputStream out = new
     FileOutputStream(outputFile);
          out.write(b);
          out.close();

//Code to read from file
        String inputFileName =
     System.getProperty("user.home",
     File.separatorChar + "home" +
     File.separatorChar + "monicap") +
     File.separatorChar + "text.txt";
          File inputFile = new File(inputFileName);
          FileInputStream in = new
     FileInputStream(inputFile);

        byte bt[] = new
     byte[(int)inputFile.length()];
        in.read(bt);
        s = new String(bt);
        in.close();
```

[1] byte streams.
[2] character streams.

```
        }catch(java.io.IOException e){
         System.out.println("Cannot access text.txt");
        }
//Clear text field
      textField.setText("");
//Display text read from file
      text.setText("Text retrieved from file:");
      textField.setText(s);
      button.setText("Click Again");
      _clickMeMode = false;
    } else {
//Save text to file
      text.setText("Text to save to file:");
      textField.setText("");
      button.setText("Click Me");
      _clickMeMode = true;
    }
  }
}
```

برای اینکه متن کاربر را در پرونده بنویسیم، متن را از textField می‌گیریم و تبدیل به یک آرایه‌ی بایتی می‌کنیم.

```
String text = textField.getText();
byte b[] = text.getBytes();
```

اینک برای پرونده‌ای که می‌خواهیم در آن بنویسیم، شیئی از نوع File ایجاد می‌کنیم، و با استفاده از آن، شیئی از نوع FileOutputStream می‌سازیم.

```
String outputFileName =
          System.getProperty("user.home",
  File.separatorChar + "home" +
  File.separatorChar + "monicap") +
  File.separatorChar + "text.txt";
  File outputFile = new File(outputFileName);
  FileOutputStream out = new
          FileOutputStream(outputFile);
```

و بالاخره، شیئ FileOutputStream آرایه‌ی بایتی را در شیئ File می‌نویسد و پس از کامل شدن عملیات، جویبار خروجی را می‌بندد.

```
out.write(b);
out.close();
```

کد مربوط به باز کردن پرونده برای خواندن نیز مشابه است. برای خواندن متن از پرونده، یک شیئ File ایجاد می‌شود، و با استفاده از آن شیئی از نوع FileInputStream ساخته می‌شود.

```
String inputFileName =
          System.getProperty("user.home",
  File.separatorChar + "home" +
  File.separatorChar + "monicap") +
  File.separatorChar + "text.txt";
  File inputFile = new File(inputFileName);
  FileInputStream out = new
          FileInputStream(inputFile);
```

اینک یک آرایه‌ی بایتی به همان اندازه‌ی پرونده ایجاد می‌شود و محتویات پرونده به درون آن خوانده می‌شود.

```
byte bt[] = new byte[(int)inputFile.length()];
in.read(bt);
```

سرانجام، با استفاده از این آرایه‌ی بایتی یک شیئ String ساخته می‌شود و متن جزء label با استفاده از آن تعیین می‌شود. پس از اتمام عملیات، FileInputStream بسته می‌شود.

```
String s = new String(bt);
```

```
label.setText(s);
in.close();
```

ویژگی‌های سیستم

در متن فوق برای تعیین مسیر مبدأ کاربر تابع `System.getProperty` فرا خوانی شده است. کلاس `System` دارای مجموعه‌ای از صفات[۱] است که خصوصیات محیط کاری فعلی را تعریف می‌کنند. هنگامی که بستر جاوا راه‌اندازی می‌شود، صفات سیستم با اطلاعات مربوط به محیط اجرا، شامل کاربر فعلی، ویراست[۲] بستر جاوا، و نویسه‌ای که برای جدا کردن اجزای نام پرونده به کار می‌رود (`File.separatorChar`)، آغازش[۳] می‌شوند.

در فرا خوانی `System.getProperty` از کلیدواژه‌ی `user.home` برای دستیابی به پوشه[۴]ی مبدأ[۵] کاربر استفاده شده است، و مقدار پیش‌فرض `File.separatorChar + "home" + File.separatorChar + "monicap"` نیز داده شده تا در صورتی که برای کلیدواژه‌ی فوق مقداری یافت نشود، از این مقدار استفاده به عمل آید.

نویسه‌ی جداکننده‌ی اجزای نام پرونده (File.separatorChar)

در متن فوق برای ساختن مسیر پوشه از متغیر `java.io.File.separatorChar` استفاده شده است. این متغیر به صورتی آغازش می‌شود که حاوی مقدار جداکننده‌ی نام پرونده باشد که در صفت سیستم `file.separator` ذخیره شده است، و با استفاده از این متغیر می‌توانید نام پورنده‌ها را به صورتی بسازید که مستقل از بستر باشند.

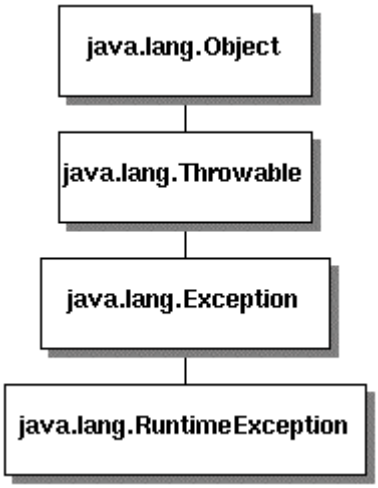
به عنوان مثال مسیر `/home/monicap/text.txt` برای یونیکس[۶] و مسیر `\home\monicap\text.txt` برای ویندوز[۷] هر دو به صورت مستقل از بستر به شکل `File.separatorChar + "home" + File.separatorChar + "monicap" + File.separatorChar + "text.txt"` نمایش داده می‌شوند.

پردازش استثناها

استثنا[۸] کلاسی است که از `java.lang.Exception` یا `java.lang.RuntimeException` منشعب می‌شود، و نشان دهنده‌ی وضعیت‌های خطایی است که برنامه‌ی شما ممکن است با آن مواجه شود. به جای اینکه در این شرایط برنامه‌ی شما بسته شود، می‌توانید متنی برای پردازش استثناها بنویسید تا برنامه به اجرای خود ادامه دهد.

---

[۱] properties.
[۲] version.
[۳] initialize.
[۴] directory.
[۵] home.
[۶] Unix.
[۷] Windows.
[۸] exception.



شکل ۱۶. استثنا

متن مربوط به ورودی و خروجی پرونده در روش `actionPerformed` در درون یک قطعه‌ی `try` و `catch` قرار گرفته تا بتوان استثنای `java.lang.IOException` را که ممکن است از متن درون قطعه افکنده شود، پردازش کرد.

`java.lang.IOException` از نوع استثناهای به اصطلاح چک شده[1] است. در بستر جاوا لازم است که یک روش تمام استثناهای چک شده‌ای را که ممکن است از متن درون آن افکنده شوند، پردازش یا اعلام کند.

استثناهای جک شده از `java.lang.Throwable` منشعب می‌شوند. اگر یک استثنای چک شده گرفته یا اعلام نشود، تدوینگر اعلام خطا می‌کند.

در این مثال، قطعه‌ی `try` و `catch` استثنای `java.lang.IOException` را می‌گیرد و پردازش می‌کند. اگر یک روش یک استثنای چک شده را نگیرد، باید امکان بر انگیخته شدن آن را اعلام کند، زیرا استثنایی که ممکن است به وسیله‌ی یک روش بر انگیخته شود، در واقع جزئی از رابط همگانی[2] آن روش است. فرا خوانندگان روش باید بدانند که روش چه استثناهایی را بر می‌انگیزد، تا اقدام لازم را انجام دهند.

اما روش `actionPerformed` رابط همگانی تعریف شده‌ای دارد که قابل تغییر نیست، بنا بر این، در این مورد، تنها کاری که باید انجام دهیم این است که استثنای چک شده را بگیریم وپردازش کنیم. روش‌هایی که شما خودتان تعریف می‌کنید، می‌توانند استثناها را اعلام یا پردازش کنند، اما روش‌هایی که جایگزین[3] می‌کنید، باید حتماً استثنا را بگیرند و پردازش کنند. در اینجا نمونه‌ای از یک روش تعریف شده توسط کاربر را می‌بینید که استثنایی را اعلام کرده است تا فرا خوانندگان بتوانند آن را بگیرند و پردازش کنند:

```
public int aComputationMethod(int number1,
 int number2)
 throws IllegalValueException{
    //Body of method
  }
```

[1] checked.
[2] public interface.
[3] override.

توجه: برای اطلاعات بیشتر در باره‌ی این موضوع به درس "پردازش خطاها و استثناها" در "خودآموز جاوا" به نشانی *http://java.sun.com/docs/books/tutorial/essential/exceptions* مراجعه کنید.

هنگامی که در برنامه‌ی خود استثنایی را می‌گیرید، باید آن را به گونه‌ای پردازش کنید که برای کاربر نهایی دوستانه باشد. کلاس‌های استثنا و خطا دارای یک روش `toString` برای چاپ متن خطا و یک روش `printStackTrace` برای چاپ وضعیت پشته[1] هستند که می‌تواند برای اشکال‌زدایی برنامه در حین تولید برنامه بسیار مفید باشد. لیکن هنگام عرضه‌ی برنامه شاید بهتر باشد که رویکرد کاربرپسندتری برای پردازش استثناها داشته باشید.

می‌توانید متن پیغام خطا را مناسب برنامه‌ی خود تغییر دهید و آن را در خط فرمان چاپ کنید یا اینکه با یک پنجره‌ی پیغام نمایش دهید. استفاده از متن مخصوص در برنامه، ترجمه‌ی برنامه به زبان‌های دیگر را هم آسان خواهد کرد، زیرا بعداً به متن خطا دسترسی خواهید داشت.

در برنامه‌های نمونه‌ی این فصل، برای پیغام خطای ورودی و خروجی پرونده متن خاصی در نظر گرفته شده است که به صورت زیر در خط فرمان چاپ می‌شود:

```
//Do this during development
  }catch(java.io.IOException e){
   System.out.println(e.toString());
   System.out.println(e.printStackTrace());
  }

//But deploy it like this
  }catch(java.io.IOException e){
   System.out.println("Cannot access text.txt");
  }
```

اگر دوست دارید که متن شما باز هم بیشتر کاربرپسند باشد، می‌توانید عملیات خواندن و نوشت را جدا کنید و ا زدو قطعه‌ی `try` و `catch` استفاده کنید. متن پیغام خطا در مورد خواندن می‌تواند به صورت *"نتوانستم پرونده‌ی text.txt را بخوانم"* و متن خطای نوشتن به صورت *"نتوانستم در پرونده‌ی text.txt را بنویسم"* باشد.

به عنوان تمرین، متن برنامه را به صورتی تغییر دهید که عملیات خواندن و نوشتن را به طور جداگانه پردازش کند. اول سعی خود را بکنید و بعد به حل آن در پرونده‌ی *FileIOError.java* (صفحه‌ی ۸۰) مراجعه نمایید.

دسترسی برنامک‌ها به پرونده‌ها

متن مربوط به دسترسی به پرونده در *FileIOAppl.java* معادل متن موجود در *FileIO.java* است، ولی چگونگی کار کردن با جویبارهای نویسه‌ای[2] را به جای جویبارهای بایتی[3] نشان می‌دهد. از هر کدام از این دو روش می‌توانید در برنامه‌ها وبرنامک‌ها استفاده کنید. در این مثال، انتخاب یکی از این دو روش کاملاً بر حسب تصادف صورت می‌گیرد، ولی در برنامه‌های واقعی این تصمیم بستگی به نیازهای خاص برنامه دارد.

تغییرات متغیر نمونه و سازنده مانند متن مربوط به برنامه هستند، و متن مربوط به روش `actionPerformed` نیز تقریباً به همان صورت است، با دو استثنا به شرح زیر:

[1] stack trace.
[2] character streams.
[3] byte streams.

- **نوشتن:** هنگامی که متن موجود در textField گرفته شد، مستقیماً به روش out.write داده می‌شود.
- **خواندن:** یک آرایه‌ی نویسه‌ای برای نگهداشتن داده‌های خوانده شده از جویبار ورودی ایجاد می‌شود.

```
public void actionPerformed(ActionEvent event){
  Object source = event.getSource();
  if(source == button){
//Variable to display text read from file
    String s = null;
    if(_clickMeMode){
      try{
//Code to write to file
        String text = textField.getText();
        String outputFileName =
                System.getProperty("user.home",
          File.separatorChar + "home" +
          File.separatorChar + "monicap") +
          File.separatorChar + "text.txt";
          File outputFile = new File(outputFileName);
          FileWriter out = new
                FileWriter(outputFile);
        out.write(text);
        out.close();
//Code to read from file
        String inputFileName =
                System.getProperty("user.home",
          File.separatorChar + "home" +
          File.separatorChar + "monicap") +
          File.separatorChar + "text.txt";
        File inputFile = new File(inputFileName);
        FileReader in = new FileReader(inputFile);
        char c[] = new
  char[(char)inputFile.length()];
        in.read(c);
        s = new String(c);
        in.close();
      }catch(java.io.IOException e){
       System.out.println("Cannot access text.txt");
      }
//Clear text field
      textField.setText("");
//Display text read from file
      text.setText("Text retrieved from file:");
      textField.setText(s);
      button.setText("Click Again");
      _clickMeMode = false;
    } else {
//Save text to file
      text.setText("Text to save to file:");
      textField.setText("");
      button.setText("Click Me");
      _clickMeMode = true;
    }
  }
}
```

اعطای اجازه به برنامک‌ها

اگر سعی کرده باشید مثال برنامک را اجرا کنید، بی‌تردید هنگام زدن دکمه‌ی **Click Me** با پیغام خطا مواجه شده‌اید. علت آن است که سیستم امنیتی بستر جاوا ۲ به یک برنامک اجازه نمی‌دهد که بدون اجازه‌ی صریح پرونده‌ای را در دیسک بنویسد یا از آن بخواند.

یک برنامک به منابع محلی سیستم دسترسی ندارد، مگر اینکه اجازه‌ی اختصاصی به آن داده شود. بنا بر این، برای اینکه برنامک `FileUIAppl` بتواند پرونده‌ی `text.txt` را بخواند یا در آن بنویسد، لازم است که اجازه‌ی دسترسی خواندنی یا نوشتنی به آن پرونده به آن داده شود.

اجازه‌ی دسترسی از طریق یک پرونده‌ی راهبرد[1] داده می‌شود، و **appletviewer** با پرونده‌ی راهبرد مربوط به برنامک مورد نظر اجرا می‌گیرد.

ایجاد یک پرونده‌ی راهبرد

ابزار راهبرد[2] یکی از ابزارهای امنیتی بستر جاوا ۲ برای ایجاد پرونده‌های راهبرد است. درس "کنترل برنامک‌ها" در خودآموز جاوا به نشانی *http://java.sun.com/docs/books/tutorial/security1.2/tour1/step2.html* نحوه‌ی استفاده از ابزار راهبرد را به طور مشروح توضیح می‌دهد. در اینجا پرونده‌ی راهبرد لازم برای اجرای برنامک را می‌بینید. برای ایجاد این پرونده می‌توانید از ابزار راهبرد استفاده کنید و ای اینکه متن زیر را به درون یک پرونده‌ی متنی کپی کنید.

```
grant {
  permission java.util.PropertyPermission
   "user.home", "read";
  permission java.io.FilePermission
   "${user.home}/text.txt", "read,write";
};
```

اجرای یک برنامک با پرونده‌ی راهبرد

به فرض اینکه پرونده‌ی راهبرد `polfile` نام داشته باشد و در همان پوشه‌ی پرونده‌ی HTML حاوی کد اجرای برنامک به نام `fileIO.html` قرار گرفته باشد، برای اجرای برنامک‌نما[3] باید از دستوری به صورت زیر استفاده کنید:

```
appletviewer -J-Djava.security.policy=polfile fileIO.html
```

> **توجه:** اگر مرورگر[4] شما مجهز به بستر جاوا ۲ باشد یا اینکه جازن جاوا (رک. نشانی *http://java.sun.com/products/plugin/index.html*) را نصب کرده باشید، می‌توانید با قرار دادن پرونده‌ی راهبرد در پوشه‌ی مبدأ رایانه‌ی خود، برنامک را از درون مرورگر اجرا کنید.

متن پرونده‌ی `fileIO.html` برای اجرای برنامک `FileIOAppl` به صورت زیر است:

```
<HTML>
<BODY>
```

[1] policy file.
[2] policy tool.
[3] appletviewer.
[4] browser.

```
<APPLET CODE=FileIOAppl.class WIDTH=200 HEIGHT=100>
</APPLET>

</BODY>
</HTML>
```

محدودسازی برنامه‌ها

برای محدود کردن دسترسی برنامه‌ها می‌توانید به صورت زیر از مدیر امنیت پیش‌فرض و یک پرونده‌ی راهبرد استفاده کنید:

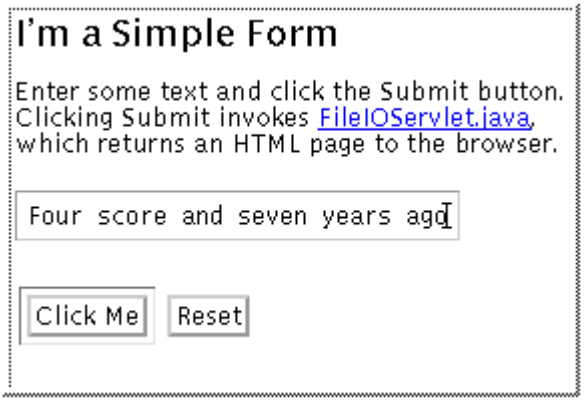
```
java -Djava.security.manager
     -Djava.security.policy=apppolfile  FileIO
```

از آنجا که برنامه در درون مدیر امنیت اجرا می‌شود، و مدیر امنیت هر گونه دسترسی را محدود می‌سازد، لذا پرونده‌ی راهبرد به دو اجازه‌ی دیگر نیاز دارد. یکی اینکه مدیر امنیت به صف رویدادها[1] دسترسی داشته باشد و اجزای رابط کاربر را بار کند، و دیگری اینکه برنامه هشداری را نمایش ندهد مبنی بر اینکه پنجره‌اش به وسیله‌ی برنامه‌ی دیگری (مدیر امنیت) ایجاد شده است.

```
grant {
  permission java.awt.AWTPermission
   "accessEventQueue";
  permission java.awt.AWTPermission
   "showWindowWithoutWarningBanner";

  permission java.util.PropertyPermission
   "user.home", "read";
  permission java.io.FilePermission
   "${user.home}/text.txt", "read,write";
};
```

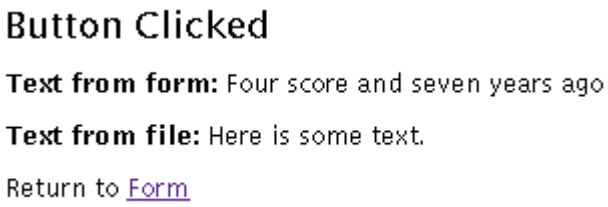
دسترسی به پرونده‌ها به وسیله‌ی سرولت‌ها

گرچه سرولت‌ها از درون مرورگر فرا خوانده می‌شوند، لیکن لیکن از نظر راهبرد امنیتی تابع مقررات سروری هستند که از آنجا اجرا می‌شوند. هنگامی که کد ورودی و خروجی پرونده را به مثال `ExampServlet.java` از فصل ۵ اضافه کنیم، می‌بینیم که تحت Java WebServer™ 1.1.1 بدون محدودیت اجرا می‌شود.



شکل ۱۷. سرولت

[1] the event queue.



شکل ۱۸. سرولت

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class FileIOServlet extends HttpServlet {

  public void doPost(HttpServletRequest request,
   HttpServletResponse response)
   throws ServletException, IOException
   {
     response.setContentType("text/html");
     PrintWriter out = response.getWriter();
     out.println("<title>Example<title>" +
                     "<body bgcolor=FFFFFF>");

     out.println("<h2>Button Clicked</h2>");

     String DATA = request.getParameter("DATA");

     if(DATA != null){
       out.println("<STRONG>Text from form:</STRONG>");
       out.println(DATA);
     } else {
       out.println("No text entered.");
     }

     try{
//Code to write to file
       String outputFileName=
         System.getProperty("user.home",
           File.separatorChar + "home" +
           File.separatorChar + "monicap") +
           File.separatorChar + "text.txt";
       File outputFile = new File(outputFileName);
       FileWriter fout = new FileWriter(outputFile);
       fout.write(DATA);
       fout.close();

//Code to read from file
       String inputFileName =
   System.getProperty("user.home",
           File.separatorChar + "home" +
           File.separatorChar + "monicap") +
           File.separatorChar + "text.txt";
       File inputFile = new File(inputFileName);
       FileReader fin = new
```

```
  FileReader(inputFile);
      char c[] = new
  char[(char)inputFile.length()];
      int i;
      i = fin.read(c);
      String s = new String(c);
      out.println("<P> <STRONG>Text from file:</STRONG>");
      out.println(s);
      fin.close();
    } catch(java.io.IOException e){
     System.out.println("Cannot access text.txt");
    }

    out.println("<P>Return to <A
  HREF=\"../simpleHTML.html\">Form</A>");
    out.close();
   }
}
```

اضافه کردن به پرونده

تا کنون در مثال‌ها دیدید که چگونه می‌توان پرونده‌ای را به طور کامل خواند یا نوشت. ولی اغلب لازم است که داده‌هایی را به یک پرونده‌ی موجود اضافه کنید یا تنها مقداری از داده‌های یک پرونده را بخوانید. با استفاده از کلاس RandomAccessFile (نشانی *http://java.sun.com/products/jdk/1.2/docs/api/java/io/RandomAccessFile.html*)،
کلاس **FileIO.java** را به گونه‌ای تغییر دهید که جمله را به آخر پرونده اضافه کند.
بعد از این که تلاش خود را کردید، جواب را در **AppendIO.java** ملاحظه کنید.

اطلاعات بیشتر

برای اطلاعات بیشتر در مورد ورودی و خروجی پرونده به درس "خواندن و نوشتن" در خودآموز جاوا به نشانی *http://java.sun.com/docs/books/tutorial/essential/io/index.html* مراجعه کنید.
در باره‌ی تعیین اندازه‌ی اجزا به قسمت "حل مسایل شایع در چینش اجزا" (*http://java.sun.com/docs/books/tutorial/ui/swingLayout/problems.html*) و "مدیریت چینش" (*http://java.sun.com/docs/books/tutorial/ui/swingOverview/layout.html*) مراجعه نمایید.

فصل ۷: دستیابی به پایگاه داده‌ای و اجازه‌ها

در این درس، مثال‌های برنامه، برنامک، و سرولت فصل ۶ را به گونه‌ای تغییر می‌دهیم که با استفاده از JDBC™ نوشتن و خواندن را در یک پایگاه داده‌ای انجام دهند. JDBC عبارت از APIهای اتصال به پایگاه داده‌ای جاوا[1] است که در بستر جاوا ۲ موجود است.

متن برنامه‌های این فصل بسیار شبیه متن برنامه‌ای است که در فصل ۶ دیدید، ولی (علاوه بر تبدیل کردن کد مربوط به دستیابی به پرونده به کد مربوط به دستیابی به پایگاه داده‌ای)، مشتمل بر مراحل دیگری، از جمله برقرار کردن محیط، ایجاد جدول پایگاه داده‌ای، و متصل شدن به پایگاه داده‌ای، نیز هست. ایجاد جدول در پایگاه داده‌ای از کارهای مدیر اجرایی پایگاه داده‌ای است و بخشی از متن برنامه‌ی شما به شمار نمی‌رود. اما متصل شدن به پایگاه داده‌ای و دستیابی حاصل از آن بخشی از برنامه‌ی شما است.

همانند فصل ۶، برنامک برای وصل شدن به پایگاه داده‌ای نیازمند اجازه‌ی مخصوص است. نوع اجازه‌ها بستگی به گرداننده[2]‌ای دارد که برای اتصال به پایگاه داده‌ای مورد استفاده قرار می‌گیرد.

برقرارسازی پایگاه داده‌ای

برای اجرای مثال‌های این فصل، نیازمند دستیابی به یک پایگاه داده‌ای هستید. می‌توانید روی سیستم خود یک پایگاه داده‌ای نصب کنید و یا شاید در محل کار دسترسی به یکپ ایگاه داده‌ای داشته باشید. در هر صورت، باید یک گردانندهٔ پایگاه داده‌ای و تنظیمات محیطی مروطه را دارا باشید تا برنامه بتواند گرداننده را بار کرده و محل پایگاه داده‌ای را پیدا کند. در ضمن برای برنامه لازم است که اطلاعات لازم را برای وارد شدن[3] به پایگاه داده‌ای (یعنی نام کاربر و گذرواژه[4]) را در اختیار داشته باشد.

گردانندهٔ پایگاه داده‌ای نرم‌افزاری است که به برنماه‌ی شما امکان می‌دهد که به یک پایگاه داده‌ای وصل شود. اگر گردانندهٔ لازم را برای پایگاه داده‌ای مورد نظر خود نداشته باشید، برنامه نخواهد توانست به آن اتصال پیدا کند.

گردانندهها یا همراه پایگاه داده‌ای عرضه می‌شوند و یا اینکه از طریق اینترنت قابل دسترسی هستند. اگر شما خود یک پایگاه داده‌ای نصب می‌کنید، برای اطلاع از چگونکی نصب و تنظیمات محیطی مربوطه به مستندات گردانندهٔ مورد نظر مراجعه کنید. اگر در محل کار از یک پایگاه داده‌ای استفاده می‌کنید، به مدیر پایگاه داده‌ای مراجعه کنید و این اطلاعات را از او کسب نمایید.

برای اینکه دو راه انجام این کار را بیاموزید، مثال برنامه از گردانندهٔ `jdbc` استفاده می‌کند، مثال برنامک از گردانندهٔ `jdbc` و `jdbc.odbc` و مثال سرولت از گردانندهٔ `jdbc.odbc`. تمام مثال‌ها به پایگاه داده‌ای `OracleOCI7.3.4` متصل می‌شوند.

---

[1] Java database connectivity.
[2] driver.
[3] login.
[4] password.

وصل شدن به پایگاه‌های داده‌ای دیگر هم مشتمل بر مراحل و متن برنامه‌ی مشابهی است. در صورتی که برای اتصال به پایگاه داده‌ای نیازمند کمک هستید، حتماً به مستندات سیستم و یا مدیر پایگاه داده‌ای مراجعه نمایید.

ایجاد جدول پایگاه داده‌ای

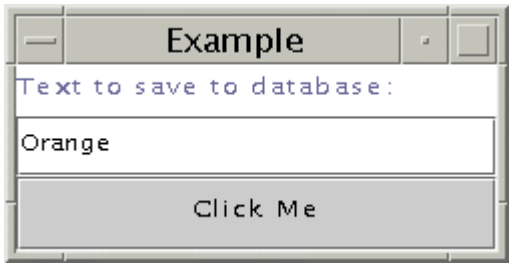
وقتی که به یک پایگاه داده‌ای دسترسی پدای کردید، برای مثال‌های این فصل جدولی در آن ایجاد کنید. جدول شما باید یک فیلد متنی برای ذخیره کردن داده‌های نویسه‌ای داشته باشد.

```
TABLE DBA (
        TEXT varchar2(100),
        primary key (TEXT)
)
```

دستیابی برنامه‌ها به پایگاه داده‌ای

در این مثال، برنامه‌ی **FileIO.java** از فصل ۶ را به گونه‌ای تغییر می‌دهیم که خواندن و نوشتن را در یک پایگاه داده‌ای انجام دهد. پنجره‌ی بالایی در شکل زیر زمانی ظاهر می‌شود که برنامه‌ی **Dba.java** را اجرا می‌کنید، و پنجره‌ی زیر آن هنگامی ظاهر می‌شود که دکمه‌ی **Click Me** را بزنید.

هنگامی که دکمه‌ی **Click Me** را می‌زنید، هر متنی که در فیلد متنی وارد کرده باشید، در پایگاه داده‌ای ذخیره می‌شود. سپس، آن داده از پایگاه داده‌ای گرفته شده و در پنجره‌ای که در پایین نشان داده شده است، نمایش داده می‌شود. اگر بیش از یک بار داده‌ها را در جدول بنویسید، تمام مطالب نوشته شده در پنجره‌ی پایینی نمایش داده می‌شود، لذا شاید لازم باشد که پنجره را بزرگ کنید تا لیست کامل موارد موجود در جدول را ببینید.



شکل ۱۹. هنگام شروع برنامه

شکل ۲۰. پس از نوشتن Orange و Apple در پایگاه داده‌ای

اجازه‌ی دسترسی به پایگاه داده‌ای مستلزم نوشتن کد برای برقراری اتصال و انجام عملیات نوشتن و خواندن از پایگاه داده‌ای است.

برقراری اتصال به پایگاه داده‌ای

کلاس `DriverManager` از JDBC می‌تواند با گردانه[1]های متعدد پایگاه داده‌ای کار کند، و همه‌ی ارتباطات پایگاه داده‌ای را آغازش می‌کند. برای بار کردن گردانه و وصل شدن به پایگاه داده‌ای، برنامه نیازمند شیئی از نوع `Connection` و رشته‌هایی برای `_driver` و `_url` است.

رشته‌ی `_url` به صورت یک جایاب متحدالشکل منبع[2] (URL) است. این رشته متشکل از URL، زیرپروتکل اوراکل[3]، و منبع داده‌ای اوراکل[4] به صورت `jdbc:oracle:thin`، `username` برای ورود[5] به پایگاه داده‌ای، `password`، به اضافه‌ی اطلاعات مربوط به دستگاه، درگاه[6]، و پروتکل است.

```
private Connection c;

final static private String _driver =
  "oracle.jdbc.driver.OracleDriver";

final static private String _url =
 "jdbc:oracle:thin:username/password@(description=(
 address_list=(address=(protocol=tcp)
 (host=developer)(port=1521)))
 (source_route=yes)(connect_data=(sid=jdcsid)))";
```

روش `actionPerformed` روش `Class.forName(_driver)` را برای بار کردن گردانه و روش `DriverManager.getConnection` را برای برقراری ارتباط فرا خوانی می‌کند. قطعه‌های try و catch در قمست پردازش استثناها در فصل ۶ مورد بحث قرار گرفته‌اند. تنها تفاوت در این است که در این بلوک از از دو دستورالعمل catch استفاده شده است، زیرا دو نوع خطا امکان‌پذیر است.

فرا خوانی `Class.forName(_driver);` استثنای `java.lang.ClassNotFoundException` را بر می‌انگیزد، و فرا خوانی `c = DriverManager.getConnection(_url);` استثنای `java.sql.SQLException` را بر

[1] driver.
[2] Uniform Resource Locator (URL).
[3] Oracle subprotcol.
[4] Oracle data source.
[5] login.
[6] port.

می‌انگیزد. در صورت بروز هر یک از این دو خطا، برنامه مسئله را به کاربر اطلاع می‌دهد، و خارج می‌شود، زیرا در صورت فقدان گردانه یا ارتباط پایگاه داده‌ای نمی‌تواند کار مفیدی انجام دهد.

```
public void actionPerformed(ActionEvent event){
  try{
//Load the driver
     Class.forName(_driver);
//Establish database connection
     c = DriverManager.getConnection(_url);
   }catch (java.lang.ClassNotFoundException e){
     System.out.println("Cannot find driver class");
     System.exit(1);
   }catch (java.sql.SQLException e){
     System.out.println("Cannot get connection");
     System.exit(1);
   }
```

متغیرهای نهایی و خصوصی

متغیرهای عضو مورد استفاده برای برقراری ارتباط با پایگاه داده‌ای در این برنامه `private` تعریف شده‌اند و در مورد دو تا از این متغیرها از عبارت `final` نیز استفاده شده است.

- **final:** یک متغیر نهایی حاوی مقدار ثابتی است که پس از آغازش به هیچ وجه قابل تغییر نیست. در این مثال، نام کاربر و گذرواژه، متغیرهای `final` هستند، زیرا نمی‌خواهیم که نمونه‌ای از این کلاس یا هر کلاس دیگر بتواند این اطلاعات را تغییر دهد.
- **private:** یک متغیر خصوصی فقط به وسیله‌ی کلاسی که آن را تعریف کرده است، قابل استفاده است. هیچ کلاس دیگری نمی‌تواند متغیرهای خصوصی را بخواند یا تغییر دهد. در این مثال، گردانه‌ی پایگاه داده‌ای، نام کاربر، و گذرواژه خصوصی اعلام شده‌اند تا یک کلاس خارجی نتواند به انها دستیابی پیدا کند و ارتباط پایگاه داده‌ای را در معرض خطر قرار دهد، و یا اینکه اطلاعات محرمانه در مورد نام کاربر و گذرواژه افشا شود. برای کسب آگاهی بیشتر د رمورد این موضوع، به درس شیئها و کلاس‌ها (به نشانی *http://java.sun.com/docs/books/tutorial/java/javaOO/index.html*) در خودآموز جاوا (*http://java.sun.com/docs/books/tutorial*) مراجعه نمایید.

خواندن و نوشتن داده‌ها

در عمل نوشتن، با استفاده از شیئ `Connection` یک شیئ `Statement` ایجاد می‌شود. شیئ `Statement` روش‌هایی برای اجرای پرس و جوها و بازسازی‌های SQL دارد. سپس، یک رشته که حاوی بازسازی SQL برای عمل نوشتن است، ساخته شده و به روش `executeUpdate` شیئ `Statement` داده می‌شود.

```
Object source = event.getSource();
if(source == button){
 JTextArea displayText = new JTextArea();

 try {
//Code to write to database
   String theText = textField.getText();
   Statement stmt = c.createStatement();
```

```
String updateString = "INSERT INTO dba VALUES
        ('" + theText + "')";
int count = stmt.executeUpdate(updateString);
```

فرمان‌های SQL، شیئهایی از نوع String هستند، و بنا بر این، ساختن آنها مانند هر رشته‌ی دیگری است؛ بدین معنا که رشته‌های ثابت در درون علاوه بر این، می‌توان نقل قول (" ") قرار می‌گیرند، و متغیرهای رشته‌ای با استفاده از علامت بعلاوه (+) به آن اضافه می‌شوند. متغیر theText دارای علامت نقل قول منفرد و دوتایی است، تا به پایگاه داده‌ای بگوید که رشته‌ی SQL دارای اطلاعات متغیر است، نه اطلاعات ثابت.

در عمل خواندن، با فرا خوانی روش executeQuery شیئ Statement، یک شیئ ResultSet ایجاد می‌شود. ResultSet حاوی داده‌هایی است که به وسیله‌ی پرس و جو بازگردانده شده است. برای بازیابی داده‌های بازگردانده شده، برنامه به طور تکراری از ResultSet داده‌ها را می‌گیرد و به ناحیه‌ی متنی displayText اضافه می‌کند.

```
//Code to read from database
    ResultSet results = stmt.executeQuery(
  "SELECT TEXT FROM dba ");
    while(results.next()){
      String s = results.getString("TEXT");
      displayText.append(s + "\n");
    }
    stmt.close();
  } catch(java.sql.SQLException e){
    System.out.println(e.toString());
  }
//Display text read from database
  panel.removeAll();
  panel.add("North", clicked);
  panel.add("Center", displayText);
  panel.add("South", clickButton);
  panel.validate();
  panel.repaint();
}
```

دستیابی به پایگاه‌های داده‌ای به وسیله‌ی برنامک‌ها

نسخه‌ی برنامکی این مثال تفاوتی با نسخه‌ی برنامه‌ای آن ندارد، جز از نظر تفاوت‌های استانداردی که بین برنامک‌ها و برنامه‌ها وجود دارد که در قسمت " ساختار و عناصر" فصل ۲ توضیح داده شد.

با این حال، اگر برنامک را بدون یک پرونده‌ی راهبرد[1] اجرا کنید، با استثنایی مواجه خواهید شد که نشان دهنده‌ی خطا در اجازه‌ی دستیابی است. در قسمت " اعطای اجازه به برنامک‌ها" از فصل ۶ کلیاتی را در باره‌ی پرونده‌های راهبرد و چگونگی اجرای برنامک با اجازه‌ی مورد نیاز برای آن بیان کردیم. در آن فصل، پرونده‌ی راهبرد را ارائه کردیم و چگونگی اجرای برنامک را به همراه آن توضیح دادیم. در این درس خواهید دید که چگونه پیغام پشته را بخوانید و بر اساس آن، اجازه‌ی مورد نیاز در پرونده‌ی راهبرد را مشخص نمایید.

برای اینکه این مثال جالب‌تر باشد، در این فصل دو برنامک برای دستیابی به پایگاه داده‌ای ارائه می‌کنیم: یکی از گردانه‌ی JDBC استفاده می‌کند، و دیگری از پل JDBC-ODBC و گردانه‌ی ODBC بهره می‌گیرد.

[1] policy file.

هر دو برنامک عملیات یکسانی را بر روی یک پایگاه داده‌ای یکسان ولی با استفاده از گردانه‌های متفاوت انجام می‌دهند. ر برنامک پرونده‌ی راهبرد مجزایی با لیست مجوزهای مخصوص خود دارد، و نیازهای متفاوتی برای جایابی گردانه‌ی پایگاه داده‌ای دارد.

گردانه‌ی JDBC

از گردانه‌های JDBC برنامه‌هایی استفاده می‌کنند که منحصراً به زان جاوا نوشته شده‌اند (برنامه‌های جاوا). این گردانه، فرا خوانی‌های جدبس را مستقیماً به پروتکل مورد استفاده‌ی DBMS تبدیل می‌کند. این گردانه توسط فروشنده‌ی DBMS عرضه می‌شود، و معمولاً به همراه بسته‌ی نرم‌افزاری DBMS در اختیار قرار می‌گیرد.

شروع کردن برنامک

برنامک DbaAppl.java برای اجرای موفق خود نیازمند یک گردانه‌ی پایگاه داده‌ای و یک پرونده‌ی راهبرد است. در این قسمت، بر پا کردن این تنظیمات را به طور کامل تشریح خواهیم کرد. پرونده‌ی **DbaAppl.html** برای اجرای برنامک `DbaAppl` به صورت زیر است:

```
<HTML>
<BODY>

<APPLET CODE=DbaAppl.class
  WIDTH=200
  HEIGHT=100>
</APPLET>

</BODY>
</HTML>
```

و دستور اجرای برنامک با **appletviewer** نیز بدین صورت است:

```
appletviewer DbaAppl.html
```

پیدا کردن محل گردانه‌ی پایگاه داده‌ای

در صورتی که گردانه به هر دلیل برای `DriverManager` قابل دسترسی نباشد، با فشار دادن دکمه‌ی **Click Me**، خطای زیر ایجاد می‌شود:

```
cannot find driver
```

معنای این خطا آن است که `DriverManager` در پوشه‌ای که پرونده‌ی HTML و پرونده‌های کلاس قرار دارند، به دنبال گردانه گشته است، و نتوانسته آن را پیدا کند. برای اصلاح این خطا، گردانه را به پوشه‌ای که پرونده‌های برنامک در آن قرار دارند، کپی کنید، و در صورتی که گردانه در درون یک پرونده‌ی زیپ شده قرار دارد، آن را از زیپ خارج کنید تا برنامک بتواند به گردانه دسترسی پیدا کند.

وقتی که گردانه در محل قرار گرفت، دوباره برنامک را اجرا کنید.

```
appletviewer DbaAppl.html
```

خواندن وضعیت پشته

با فرض اینکه گردانه به طور محلی در دسترس برنامک قرار داشته باشد، اگر برنامک DbaAppl.java بدون یک پرونده‌ی راهبرد اجرا شود، با فشار دادن دکمه‌ی **Click Me**، خطای زیر ایجاد می‌شود:

```
java.security.AccessControlException: access denied
```

```
(java.net.SocketPermission developer resolve)
```

اولین سطر خطای فوق می‌گوید که اجازه‌ی دستیابی داده نشده است. به عبارت دیگر، علت ایجاد خطا آن است که برنامک سعی کرده است بدون داشتن اجازه‌ی لازم به منابع سیستم دستیابی پیدا کند. سطر دوم بدان معنا است که برای اصلاح این وضعیت، یک `SocketPermission` لازم است که به برنامک اجازه‌ی دستسابی به دستگاهی که پایگاه داده‌ای روی آن واقع شده است (`developer`)، بدهد.

برای ایجاد پرونده‌ی راهبرد لازم می‌توانید از ابزار راهبرد استفاده کنید و ای اینکه می‌توانید با استفاده از یک ویرایشگر اسکی[1] آن را به صورت زیر تولید نمایید. پرونده‌ی راهبرد مورد نظر که اجازه‌ی اخیر در آن قرار گرفته است، در زیر نشان داده شده است:

```
grant {
  permission java.net.SocketPermission "developer",
    "resolve";
  "accessClassInPackage.sun.jdbc.odbc";
};
```

یک بار دیگر برنامک را اجرا کنید، و این بار پرونده‌ی راهبرد `DbaApplPol` را که حاوی اجازه‌ی فوق است، در آن قرار دهید:

```
appletviewer -J-Djava.security.policy=DbaApplPol
                                        DbaAppl.html
```

باز دوباره یک خطا دریافت می‌کنید، ولی این بار از نوع متفاوتی است:

```
  java.security.AccessControlException: access denied
  (java.net.SocketPermission
  129.144.176.176:1521 connect,resolve)
```

حالا نیاز به اجازه‌ای از نوع `SocketPermission` دارید که اجازه‌ی دسیابی به نشانی و درگاه پروتکل اینترنت (IP) روی ماشین **developer** را که پایگاه داده‌ای در آن واقع شده است، بدهد.

در اینجا پرونده‌ی راهبرد **DbaApplPol** را که اجازه‌ی اخیر نیز به آن اضاغفه شده است، می‌بینید:

```
grant {
  permission java.net.SocketPermission "developer",
                        "resolve";
  permission java.net.SocketPermission
  "129.144.176.176:1521", "connect,resolve";
};
```

دوباره برنامک را اجرا کنید. اگر از پرونده‌ی راهبرد فوق استفاده کنید که اجازه‌ی سوکت نیز در آن درج شده است، مشکلی نخواهید داشت.

```
  appletviewer -J-Djava.security.policy=DbaApplPol
                                        DbaAppl.html
```

پل JDBC-ODBC با گردانه‌ی ODBC

ODBC[2] یک رابط برنامه‌نویسی است که میکروسافت برای دستیابی به تعداد زیادی از پایگاه‌های داده‌ای رابطه‌ای در بسترهای مختلف ابداع کرده است. پل JDBC-ODBC در نسخه‌های جاوای سولاریس و ویندوز قرار داده شده است، و دو امکان مهم را در اختیار شما قرار می‌دهد:

۱. استفاده از ODBC از درون یک برنامه‌ی جاوا

[1] ASCII editor.
[2] Open DataBase Connectivity (ODBC).

۲. بار کردن گردانه‌های ODBC به عنوان گردانه‌ی JDBC. در این مثال از پل JDBC-ODBC برای وصل شدن به یک پایگاه داده‌ای استفاده می‌شود. با این حال، هیچگونه کد ODBC در این برنامک به کار نرفته است.

DriverManager برای یافتن محل گردانه‌های پایگاه داده‌ای و بار کردن آنها از متغیرهای محیطی استفاده می‌کند. برای این مثال، لازم نیست که پرونده‌ی گردانه به طور محلی در دسترس باشد.

شروع برنامک

پرونده‌ی **DbaOdb.html** برای اجرای برنامک DbaOdbAppl به شرح زیر است:

```
<HTML>
<BODY>

<APPLET CODE=DbaOdbAppl.class
  WIDTH=200
  HEIGHT=100>
</APPLET>

</BODY>
</HTML>
```

و چگونگی اجرای آن به صورت زیر است:

```
appletviewer DbaOdb.html
```

خواندن جریان پشته

اگر برنامک DbaOdbAppl.java بدون پرونده‌ی راهبرد اجرا شود، با فشار دادن دکمه‌ی **Click Me** توسط کاربر، پیغام پشته‌ی زیر ایجاد می‌شود:

```
java.security.AccessControlException: access denied
(java.lang.RuntimePermission
accessClassInPackage.sun.jdbc.odbc )
```

اولین سطر این پیغام به ما می‌گوید که اجازه‌ی دسترسی داده نشد. این بدان معنا است که علت بروز این پیغام خطا این است که برنامک سعی کرده است بدون داشتن اجازه‌ی لازم به یکی از منابع سیستم دست پیدا کند. سطر دوم بدان معنا است که برنامک باید اجازه‌ای از نوع RuntimePermission برای دستیابی به بسته‌ی sun.jdbc.odbc داشته باشد. این بسته، کار پل JDBC-ODBC را برای ماشین مجازی جاوا انجام می‌دهد.

برای ایجاد پرونده‌ی راهبرد مورد نیاز می‌توانید از ابزار راهبرد استفاده کنید، و یا اینکه می‌توانید محتویات زیر را با استفاده از یک ویرایشگر اسکی (ASCII) در پرونده بنویسید. در اینجا پرونده‌ی راهبرد مورد نظر را می‌بینید که اجازه‌ی مورد نیاز در آن داده شده است:

```
grant {
  permission java.lang.RuntimePermission
    "accessClassInPackage.sun.jdbc.odbc";
};
```

بار دیگر برنامک را اجرا کنید، و این بار پرونده‌ی **DbaOdbPol** را که حاوی اجازه‌ی مذکور است، به عنوان پرونده‌ی راهبرد به آن بدهید:

```
appletviewer -J-Djava.security.policy=DbaOdbPol DbaOdb.html
```

باز هم یک پیغام پشته دریافت می‌کنید، ولی این بار وضعیت خطای آن متفاوت است:

```
java.security.AccessControlException:
```

```
access denied (java.lang.RuntimePermission
file.encoding read)
```

این پیغام پشته بدان معنا است که برنامک نیازمند اجازه‌ی خواندن برای پرونده کد شده (دودویی) است. متن پرونده‌ی **DbaOdbPol** که این اجازه نیز به آن اضافه شده است، در زیر مشاهده می‌شود:

```
grant {
  permission java.lang.RuntimePermission
     "accessClassInPackage.sun.jdbc.odbc";
  permission java.util.PropertyPermission
     "file.encoding", "read";
};
```

بار دیگر برنامک را اجرا کنید. اگر از پرونده‌ی راهبرد فوق که دارای اجازه‌ی Runtime و Property است، استفاده کنید، برنامک اجرا می‌شود.

```
appletviewer -J-Djava.security.policy=DbaOdbPol DbaOdb.html
```

دستیابی به پایگاه داده‌ای توسط سرولت‌ها

به طوری که در فصل ۶ یاد گرفتید، سرولت‌ها تحت راهبرد امنیتی سرور شبکه‌ای هستند که در آن اجرا می‌شوند. هنگامی که کد خواندن و نوشتن پایگاه داده‌ای به `FileIOServlet` مربوط به فصل ۶ افزوده شود، سرولت این فصل (DbaServlet.java) حاصل می‌شود که تحت Java WebServer™ 1.1.1 بدون هر گونه محدودیتی اجرا می‌شود.

سرور شبکه را باید به گونه پیکربندی کرد که بتواند محل پایگاه داده‌ای را پیدا کند. برای کمک در این زمینه به مستندات سرور شبکه یا مدیر پایگاه داده‌ای خود مراجعه کنید. در مورد Java WebServer™ 1.1.1، تنظیم پیکربندی شامل ویرایش نوشتارهای شروع و تغییر دادن تنظیمات محیطی برای بار کردن گردانه‌ی ODBC و یافتن جای پایگاه داده‌ای و وصل شدن به آن است.

شکل ۲۱.

شکل ۲۲.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

import java.sql.*;
import java.net.*;
import java.io.*;

public class DbaServlet extends HttpServlet {

 private Connection c;
 final static private String _driver =
  "sun.jdbc.odbc.JdbcOdbcDriver";
 final static private String _user = "username";
 final static private String _pass = "password";
 final static private String
       _url = "jdbc:odbc:jdc";

public void doPost(HttpServletRequest request,
   HttpServletResponse response)
   throws ServletException, IOException{
  response.setContentType("text/html");
  PrintWriter out = response.getWriter();
  out.println("<title>Example<title>" +
      "<body bgcolor=FFFFFF>");

  out.println("<h2>Button Clicked</h2>");

  String DATA = request.getParameter("DATA");

  if(DATA != null){
     out.println("<STRONG>Text from
       form:</STRONG>");
     out.println(DATA);
  } else {
    out.println("No text entered.");
  }

//Establish database connection
  try{
    Class.forName (_driver);
    c = DriverManager.getConnection(_url,
              _user,
              _pass);
```

```
    } catch (Exception e) {
      e.printStackTrace();
      System.exit(1);
    }

    try{
//Code to write to database
      Statement stmt = c.createStatement();
      String updateString = "INSERT INTO dba " +
      "VALUES ('" + DATA + "')";
      int count = stmt.executeUpdate(updateString);

//Code to read from database
      ResultSet results = stmt.executeQuery(
      "SELECT TEXT FROM dba ");
      while(results.next()){
        String s = results.getString("TEXT");
        out.println("<BR>
          <STRONG>Text from database:</STRONG>");
        out.println(s);
      }
       stmt.close();
      }catch(java.sql.SQLException e){
       System.out.println(e.toString());
      }

       out.println("<P>Return to
         <A HREF="../dbaHTML.html">Form</A>");
       out.close();
  }
}
```

اطلاعات بیشتر

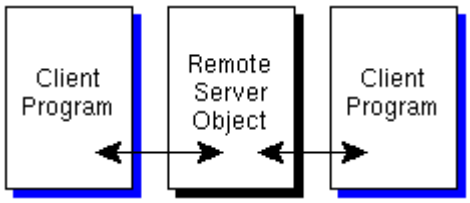
برای اطلاعات بیشتر در باره‌ی شرایط دستیابی به متغیرها به درس "شیئها و کلاس‌ها" (به نشانی *http://java.sun.com/docs/books/tutorial/java/javaOO/index.html*) در "خودآموز جاوا" (*http://java.sun.com/docs/books/tutorial*) مراجعه نمایید.

فصل ۸: فرا خوانی دوردست روش

API فرا خوانی دوردست روش (RMI) در جاوا ارتباط مشتری و سرور را بر روی شبکه امکان‌پذیر می‌سازد. به طور معمول، برنامه‌های مشتری تقاضاهایی را برای برنامه‌ی سرور می‌فرستند، و برنامه‌ی سرور به آن تقاضاها پاسخ می‌دهد.

مثال شایع این روش، استفاده‌ی اشتراکی از یک واژه‌پرداز بر روی شبکه است. واژه‌پرداز بر روی سرور نصب می‌شود، و هر کس می‌خواهد از آن استفاده کند، با دوکلیک کردن بر روی آیکونی در میز کار رایانه‌ی خود و یا تایپ کردن دستور در خط فرمان آن را اجرا می‌نماید. برنمه‌ی مشتری تقاضایی را برای دستیابی به نرم‌افزار برای برنامه‌ی سرور می‌فرستد، و برنامه‌ی سرور با قرار دادن نرم‌افزار در اختیار مشتری به این تقاضا پاسخ می‌دهد.

RMI API به شما امکان می‌دهد که روی سرور شیئی ایجاد کنید که از دوردست برای مشتریان قابل دسترس است، و از طریق فرا خوانی روش‌ها روی این شیئ، ارتباط سرور و مشتری امکان‌پذیر می‌شود. مشتریان به آسانی می‌توانند مستقیماً با شیئ سرور ارتباط برقرار کنند، و از طریق این شیئ، به طور غیرمستقیم، با استفاده از جایاب‌های یکنواخت منابع[1] (URLها) و پروتکل انتقال ابرمتن[2] (HTTP)، با یکدیگر نیز ارتباط برقرار نمایند.



شکل ۲۳.

این فصل چگونگی استفاده از RMI API را برای برقراری ارتباط سرور و مشتری توضیح می‌دهد.

در باره‌ی مثال

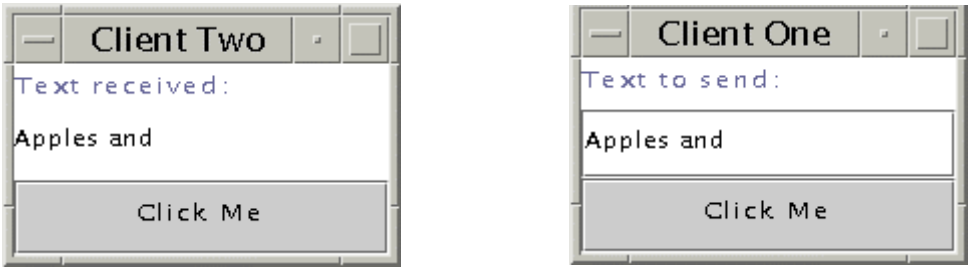
در این فصل، برنامه‌ی ورودی و خروجی پرونده (FileIO.java) از فصل ۶ (**دستیابی به پرونده و اجازه‌ها**) را به RMI API تبدیل خواهیم کرد.

رفتار برنامه

برنامه‌ی RMIClient1.java رابط کاربر ساده‌ای دارد و از کاربر ورودی متن می‌خواهد. وقتی که دکمه‌ی **Click Me** را می‌زنید، متن از طریق شیئ سرور دوردست به برنامه‌ی RMIClient2.java فرستاده می‌شود.

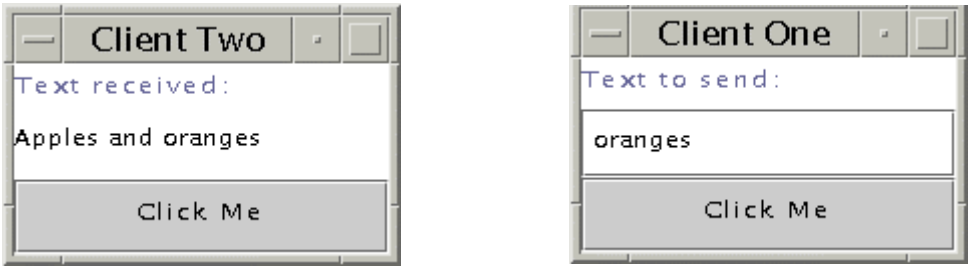
[1] Uniform Resource Locators (URLs).
[2] HyperText Transfer Protocol (HTTP).

هنگامی که در برنامه‌ی RMIClient2.java دکمه‌ی **Click Me** را می‌زنید، متن فرستاده شده از RMIClient1.java ظاهر می‌شود.



شکل ۲۴. نمونه‌ی اول مشتری ۱

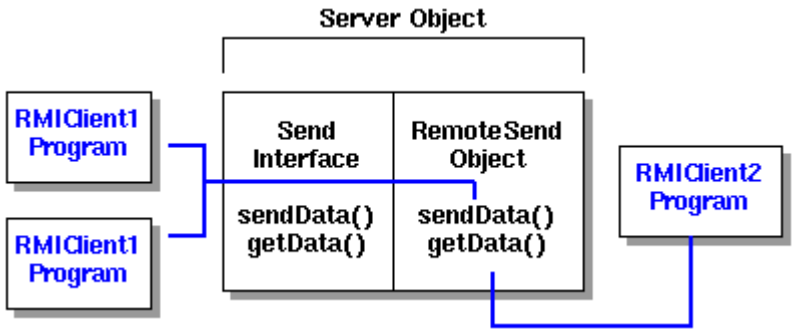
اگر نمونه‌ی دومی از `RMIClient1` را اجرا کنید و متنی را در آن تایپ نمایید، وقتی دکمه‌ی **Click Me** را بزنید، آن متن برای `RMIClient2` فرستاده می‌شود. برای دیدن این متن در برنامه‌ی اخیر، دکمه‌ی **Click Me** را بزنید.



شکل ۲۵. نمونه‌ی دوم مشتری ۱

خلاصه‌ی پرونده‌ها

برنامه‌ی مثال ما متشکل از برنامه‌ی `RMIClient1`، شیئ و رابط دوردست، و برنامه‌ی `RMIClient2` است، که ارتباطات آنها در شکل زیر نشان داده شده است. پرونده‌های متن مربوط به این برنامه‌ها در لیست زیر نام برده شده‌اند.



شکل ۲۶.

- RMIClient1.java: برنامه‌ی مشتری که روش sendData را روی شیئ سرور RemoteServer فرا خوانی می‌کند.
- RMIClient2.java: برنامه‌ی مشتری که روش getData را روی شیئ سرور RemoteServer فرا خوانی می‌کند.
- RemoteServer.java: شیئ دوردست سرور که رابط Send.java و روش‌های sendData و getData را پیاده‌سازی می‌کند.
- Send.java: رابط دوردستی که روش‌های سرور دوردست sendData و getData را اعلام می‌کند.

علاوه بر این، پرونده‌ی راهبرد امنیتی java.policy زیر نیز اجازه‌های لازم را برای اجرای مثال می‌دهد:

```
grant {
  permission java.net.SocketPermission
              "*:1024-65535",
              "connect,accept,resolve";
  permission java.net.SocketPermission
              "*:80", "connect";
  permission java.awt.AWTPermission
              "accessEventQueue";
  permission java.awt.AWTPermission
              "showWindowWithoutWarningBanner";
};
```

تدوین کردن مثال

در دستورالعمل‌های زیر، فرض بر این است که برنامه در شاخه‌ی آغازین کاربر zelda قرار دارد. برنامه‌ی سرور در شاخه‌ی آغازین کاربر zelda تدوین می‌شود، لیکن برای اجرا به شاخه‌ی public_html مربوط به کاربر zelda برده می‌شود.

مجموعه‌ی فرمان‌ها برای بسترهای یونیکس و ویندوز ۳۲ بیتی به شرح زیر است—توضیحات در زیر ارائه خواهد شد:

Unix:

```
cd /home/zelda/classes
javac Send.java
javac RemoteServer.java
javac RMIClient2.java
javac RMIClient1.java
rmic -d . RemoteServer
cp RemoteServer*.class /home/zelda/public_html/classes
cp Send.class /home/zelda/public_html/classes
```

Win32:

```
cd \home\zelda\classes
javac Send.java
javac RemoteServer.java
javac RMIClient2.java
javac RMIClient1.java
rmic -d . RemoteServer
copy RemoteServer*.class \home\zelda\public_html\classes
copy Send.class \home\zelda\public_html\classes
```

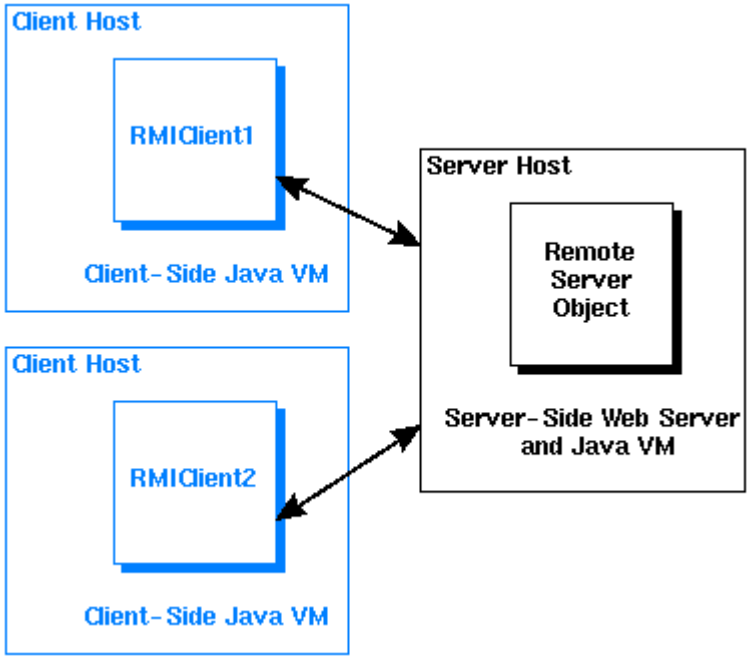
دو فرمان javac اول، کلاس و رابط RemoteServer و Send را تدوین می‌کنند. فرمان javac سوم کلاس RMIClient2 را تدوین می‌نماید. آخرین فرمان javac، کلاس RMIClient1 را تدوین می‌کند.

سطر بعدی فرمان rmic را روی کلاس سرور RemoteServer اجرا می‌نماید. این فرمان پرونده‌های کلاس از نوع ClassName_Stub.class و ClassName_Skel.class ایجاد می‌کند. این کلاس‌ها امکان آن را فراهم می‌کنند که مشتری‌ها روش‌های شیئ سرور RemoteServer را فرا خوانی کنند.

اولین فرمان کپی پرونده‌ی کلاس RemoteServer و کلاس‌های skel و stub مربوط به آن را به شاخه‌ی عمومی /home/zelda/public_html/classes منتقل می‌کند که روی سرور قرار دارد و لذا همه می‌توانند آنها را فروگذاری کنند. اینها در شاخه‌ی public_html از سرور شبکه قرار می‌گیرند، زیرا برنامه‌های مشتری با استفاده از URL به آنها دستیابی می‌کنند.

فرمان کپی دوم کلاس send را برای همان منظور به همان محل می‌فرستد. کلاس‌های RMIClient1 و RMIClient2 در دسترس عموم قرار نمی‌گیرند؛ این کلاس‌ها با استفاده از URL با مشتری برای دستیابی و فروگذاری پرونده‌های شیئ دوردست در شاخه‌ی public_html ارتباط برقرار می‌کنند.

- RMIClient1 از یک شاخه‌ی مشتری فرا خوانده می‌شود، و با استفاده از سرور شبکه در سمت سرور و ماشین مجازی جاوا در سمت مشتری پرونده‌های عمومی موجود را فروگذاری می‌کند.
- RMIClient2 از یک شاخه‌ی مشتری فرا خوانده می‌شود، و با استفاده از سرور شبکه در سمت سرور و ماشین مجازی جاوا در سمت مشتری پرونده‌های عمومی موجود را فروگذاری می‌کند.



شکل ۲۷.

راه‌اندازی رجیستری RMI

پیش از شروع برنامه‌های مشتری، باید رجیستری RMI را راه‌اندازی کنید، که یک مخزن نامگذاری سمت سرور است که به مشتری‌های دوردست امکان می‌دهد که اشاره‌ای به شیئ سرور دوردست به دست آورند.

پیش از راه‌اندازی رجیستری RMI، دقت کنید که پوسته یا پنجره‌ای که فرمان rmiregistry در آن اجرا می‌شود، دارای یک متغیر محیطی CLASSPATH نباشد که به محل کلاس‌های شیئ دوردست، از جمله کلاس‌های stub و skel در روی سیستم اشاره داشته باشد. اگر رجیستری RMI در زمان اجرا این کلاس‌ها را پیدا کند، آنها را از ماشین مجازی جاوای سمت سرور بار نخواهد کرد، که این زمانی که مشتری‌ها بخواهند کلاس‌های سرور دوردست را فروگذاری کنند، باعث مشکل خواهد شد.

فرمان‌های زیر، مقدار متغیر CLASSPATH را حذف می‌کنند، و رجیستری RMI را روی درگاه پیش‌فرض ۱۰۹۹ راه‌اندازی می‌نمایند. می‌توانید با اضافه کردن شماره، درگاه دیگری به صورت زیر تعریف کنید: rmiregistry 4444 &. اگر شماره‌ی درگاه دیگری تعیین نمایید، باید همان شماره را در کد سمت سرور نیز وارد نمایید.

Unix:
```
cd /home/zelda/public_html/classes
unsetenv CLASSPATH
rmiregistry &
```
Win32:
```
cd \home\zelda\public_html\classes
set CLASSPATH=
start rmiregistry
```

توجه: بعد از اتمام این کار می‌توانید متغیر CLASSPATH را به مقدار قبلی آن باز گردانید.

اجرای شیئ سرور RemoteServer

برای اجرای برنامه‌های مثال، ابتدا RemoteServer را اجرا کنید. اگر ابتدا RMIClient1 یا RMIClient2 را اجرا نمایید، نخواهند توانست ارتباط برقرار کنند، زیرا شیئ سرور دوردست در حال اجرا نیست.

در این مثال، RemoteServer از شاخه‌ی /home/zelda/public_html/classes اجرا می‌شود.

فرمان java تا آخر در یک سطر نوشته می‌شود و به جای آخر خط[1] فضای خالی[2] گذاشته می‌شود. صفاتی که با گزینه‌ی D- برای تفسیرگر java تعیین شده است، خصوصیاتی است که رفتار برنامه را تغییر می‌دهد.

Unix:
```
cd /home/zelda/public_html/classes
java -Djava.rmi.server.codebase=http://kq6py/~zelda/classes
   -Djava.rmi.server.hostname=kq6py.eng.sun.com
   -Djava.security.policy=java.policy RemoteServer
```
Win32:
```
cd \home\zelda\public_html\classes
java -Djava.rmi.server.codebase=file:c:\home\zelda\public_html\classes
   -Djava.rmi.server.hostname=kq6py.eng.sun.com
   -Djava.security.policy=java.policy RemoteServer
```

- صفت java.rmi.server.codebase محل کلاس‌های در دسترس عموم را مشخص می‌کند.
- صفت java.rmi.server.hostname نام کامل میزبان سرور است که کلاس‌های در دسترس عموم در آن واقع شده‌اند.

[1] line break.
[2] space.

- صفت java.security.policy پرونده‌ی راهبرد **java.policy** را مشخص می‌کند که اجازه‌های لازم برای اجرای شیئ سرور دوردست و دستیابی به کلاس‌های سرور دوردست برای فروگذاری در آن داده شده است.
- کلاسی که باید اجرا شود (RemoteServer).

اجرای برنامه‌ی RMIClient1

مجموعه‌ی فرمان‌های لازم برای بسترهای یونیکس و ویندوز ۳۲ بیتی و توضیحات مربوطه در زیر آمده است.
در این مثال، RMIClient1 از شاخه‌ی /home/zelda/classes اجرا می‌شود.
فرمان java تا آخر در یک سطر نوشته می‌شود و به جای آخر خط[1] فضای خالی[2] گذاشته می‌شود. صفاتی که با گزینه‌ی D- برای تفسیرگر java تعیین شده است، خصوصیاتی است که رفتار برنامه را تغییر می‌دهد.

Unix:
```
cd /home/zelda/classes

java -Djava.rmi.server.codebase=http://kq6py/~zelda/classes/
  -Djava.security.policy=java.policy RMIClient1 kq6py.eng.sun.com
```
Win32:
```
cd \home\zelda\classes

java -Djava.rmi.server.codebase=file:c:\home\zelda\classes\
  -Djava.security.policy=java.policy RMIClient1 kq6py.eng.sun.com
```

- صفت java.rmi.server.codebase محل کلاس‌های در دسترس عموم را برای فروگذاری مشخص می‌کند.
- صفت java.security.policy پرونده‌ی راهبرد **java.policy** را مشخص می‌کند که اجازه‌های لازم برای برنامه‌ی مشتری جهت دستیابی به کلاس‌های سرور دوردست در آن داده شده است.
- برنامه‌ی مشتری که باید اجرا شود (RMIClient1)، و نام میزبان سروری که کلاس‌های سرور دوردست در آن واقع شده‌اند (Kq6py).

اجرای RMIClient2

مجموعه‌ی فرمان‌های لازم برای بسترهای یونیکس و ویندوز ۳۲ بیتی و توضیحات مربوطه در زیر آمده است.
در این مثال، RMIClient2 از شاخه‌ی /home/zelda/classes اجرا می‌شود.
فرمان java تا آخر در یک سطر نوشته می‌شود و به جای آخر خط[3] فضای خالی[4] گذاشته می‌شود. صفاتی که با گزینه‌ی D- برای تفسیرگر java تعیین شده است، خصوصیاتی است که رفتار برنامه را تغییر می‌دهد.

Unix:
```
cd /home/zelda/classes
java -Djava.rmi.server.codebase=http://kq6py/~zelda/classes
  -Djava.security.policy=java.policy RMIClient2 kq6py.eng.sun.com
```
Win32:
```
cd \home\zelda\classes
```

[1] line break.
[2] space.
[3] line break.
[4] space.

```
java -Djava.rmi.server.codebase=file:c:\home\zelda\public_html\classes
  -Djava.security.policy=java.policy RMIClient2 kq6py.eng.sun.com
```

- صفت java.rmi.server.codebase محل کلاس‌های در دسترس عموم را مشخص می‌کند.
- صفت java.rmi.server.hostname نام کامل میزبان سرور است که کلاس‌های در دسترس عموم در آن واقع شده‌اند.
- صفت java.security.policy پرونده‌ی راهبرد **java.policy** را مشخص می‌کند که اجازه‌های لازم برای اجرای شیئ سرور دوردست و دستیابی به کلاس‌های سرور دوردست برای فروگذاری در آن داده شده است.
- کلاسی که باید اجرا شود (RMIClient2).

کلاس RemoteServer

کلاس RemoteServer کلاس UnicastRemoteObject را گسترش می‌دهد و روش‌های sendData و getData را که در رابط Send اعلام شده‌اند، پیاده‌سازی می‌کند. کلاس UnicastRemoteObject برخی از روش‌های java.lang.Object را برای اشیای دوردست پیاده‌سازی می‌کند، و سازنده‌ها و روش‌های ایستایی دارد که باعث می‌شوند برنامه‌های مشتری بتوانند روش‌های شیئ دوردست را فرا خوانی کنند.

```
class RemoteServer extends UnicastRemoteObject
                   implements Send {

  String text;

  public RemoteServer() throws RemoteException {
    super();
  }

  public void sendData(String gotText){
    text = gotText;
  }

  public String getData(){
    return text;
  }
```

روش main، RMISecurityManager را نصب می‌کند، و ارتباطی را با یک درگاه در ماشین سرور برقرار می‌کند. مدیر امنیت مشخص می‌کند که آیا برای عملیات مورد نظر برنامه‌ی سرور که نیاز به کسب اجازه دارند، اجازه‌ی لازم از طریق پرونده‌ی راهبرد داده شده است، یا نه. روش main برای شیئ RemoteServer یک نام ایجاد می‌کند، که حاوی نام سرور (kq6py) اجرا کننده‌ی رجیستری RMI و شیئ دوردست، و نام Send است.

نام سرور به طور پیش‌فرض از درگاه ۱۰۹۹ استفاده می‌کند. اگر مایل باشید که از درگاه دیگری استفاده کنید، می‌توانید آن را با دونقطه به صورت زیر اضافه نمایید: kq6py:4444. اگر به این ترتیب درگاه را تغییر دهید، باید رجیستری RMI را با همان شماره‌ی درگاه اجرا نمایید.

در قطعه‌ی try، نمونه‌ای از کلاس RemoteServer ایجاد می‌شود، و با استفاده از دستورالعمل Naming.rebind(name, remoteServer);، نام مشخص شده به شیئ دوردست در رجیستری RMI متصل می‌شود.

```
  public static void main(String[] args){
    if(System.getSecurityManager() == null) {
      System.setSecurityManager(new
               RMISecurityManager());
    }
    String name = "//kq6py.eng.sun.com/Send";
    try {
      Send remoteServer = new RemoteServer();
      Naming.rebind(name, remoteServer);
      System.out.println("RemoteServer bound");
    } catch (java.rmi.RemoteException e) {
      System.out.println("Cannot create
                  remote server object");
    } catch (java.net.MalformedURLException e) {
      System.out.println("Cannot look up
                  server object");
    }
  }
}
```

توجه: شیئ remoteServer از نوع Send است، نه از نوع کلاس RemoteServer، زیرا تنها رابط Send و روش‌های آن در اختیار مشتریان قرار دارد، نه کلاس مذکور و روش‌های آن.

رابط Send

رابط Send روش‌هایی تعریف می‌کند که در کلاس RemoteServer پیاده‌سازی می‌شوند. اینها روش‌هایی هستند که از دوردست قابل دسترسی‌اند.

```
public interface Send extends Remote {

  public void sendData(String text)
               throws RemoteException;
  public String getData() throws RemoteException;
}
```

کلاس RMIClient1

کلاس RMIClient1 ارتباطی را با برنامه‌ی سرور دوردست برقرار می‌کند، و داده‌ها را به شیئ سرور دوردست می‌فرستد. کد انجام دهنده‌ی این کارها در روش‌های actionPerformed و main قرار دارد.

روش actionPerformed

روش actionPerformed برای ارسال متن به شیئ سرور دوردست، روش RemoteServer.sendData را فرا خوانی می‌کند.

```
public void actionPerformed(ActionEvent event){
   Object source = event.getSource();

   if(source == button){
//Send data over socket
      String text = textField.getText();
```

```
      try{
        send.sendData(text);
      } catch (java.rmi.RemoteException e) {
        System.out.println("Cannot send data to server");
      }
      textField.setText(new String(""));
    }
}
```

روش main

روش main، پس از نصب RMISecurityManager، نامی را ایجاد می‌کند و برای دسترسی به شیئ سرور دوردست از این نام استفاده می‌نماید. مشتری از روش Naming.lookup برای پیدا کردن شیئ RemoteServer در رجیستری RMI که روی سرور اجرا می‌شود، استفاده می‌نماید.
مدیر امنیت مشخص می‌کند که آیا پرونده‌ی راهبرد مناسب برای اعطای اجازه‌ی لازم به کد فروگذاری شده برای انجام کارهای مستلزم کسب اجازه وجود دارد یا نه.

```
  RMIClient1 frame = new RMIClient1();

  if(System.getSecurityManager() == null) {
    System.setSecurityManager(new RMISecurityManager());
  }

  try {
//args[0] contains name of server where Send runs
    String name = "//" + args[0] + "/Send";
    send = ((Send) Naming.lookup(name));
  } catch (java.rmi.NotBoundException e) {
    System.out.println("Cannot look up
                remote server object");
  } catch(java.rmi.RemoteException e){
    System.out.println("Cannot look up
                remote server object");
  } catch(java.net.MalformedURLException e) {
    System.out.println("Cannot look up
                remote server object");
  }
```

کلاس RMIClient2

کلاس RMIClient2 با برنامه‌ی سرور دوردست ارتباط برقرا رکرده، داده‌ها را از سرور دوردست می‌گیرد، و آن را نمایش می‌دهد. کد انجام دهنده‌ی این کارها در روش‌های actionPerformed و main قرار دارد.

روش actionPerformed

روش actionPerformed برای بازیابی داده‌های فرستاده شده توسط برنامه‌ی مشتری، روش RemoteServer.getData را فرا خوانی می‌کند. این داده‌ها به آخر متن شیئ TextArea افزوده می‌شود، تا برای کارب رنهایی در سمت سرور نمایش داده شود.

```
public void actionPerformed(ActionEvent event) {
   Object source = event.getSource();

   if(source == button){
      try{
        String text = send.getData();
```

```
            textArea.append(text);
          } catch (java.rmi.RemoteException e) {
            System.out.println("Cannot send data
                        to server");
          }
          }
      }
}
```

روش main

روش main، پس از نصب RMISecurityManager، نامی را ایجاد می‌کند و برای دسترسی به شیئ سرور دوردست از این نام استفاده می‌نماید. پارامتر args[0] نام سرور میزبان را مشخص می‌کند. مشتری از روش Naming.lookup برای پیدا کردن شیئ RemoteServer در رجیستری RMI که روی سرور اجرا می‌شود، استفاده می‌نماید.

مدیر امنیت مشخص می‌کند که آیا پرونده‌ی راهبرد مناسب برای اعطای اجازه‌ی لازم به کد فروگذاری شده برای انجام کارهای مستلزم کسب اجازه وجود دارد یا نه.

```
RMIClient2 frame = new RMIClient2();

if(System.getSecurityManager() == null) {
  System.setSecurityManager(new RMISecurityManager());
}

try {
  String name = "//" + args[0] + "/Send";
  send = ((Send) Naming.lookup(name));
} catch (java.rmi.NotBoundException e) {
  System.out.println("Cannot look up remote
              server object");
} catch(java.rmi.RemoteException e){
  System.out.println("Cannot look up remote
              server object");
} catch(java.net.MalformedURLException e) {
  System.out.println("Cannot look up remote
              server object");
}
```

اطلاعات بیشتر

برای اطلاعات بیشتر، به درس RMI (به نشانی *http://java.sun.com/docs/books/tutorial/rmi/index.html*) در "خودآموز جاوا" مراجعه کنید.

در خاتمه

پس از تکمیل این خودآموز، باید تا حدودی با زبان برنامه‌نویسی جاوا و چگونگی استفاده از رابط‌های معمول برنامه‌نویس یکاربردی در بستر جاوا آشنا شده باشید. در ضمن، باید درک روشنی از شباهت‌ها و تفاوت‌های سه نوع شایع برنامه‌های زبان جاوا — برنامه‌های کاربردی، برنامک‌ها، و سرولت‌ها — پیدا کرده باشید.

بخش دوم این کتاب به بررسی سوکت‌ها، ریسه‌ها، رمزنگاری، ساختن رابط کاربر پیچیده‌تر، متوالی‌سازی، انباره‌ها، بین‌المللی‌سازی، و پرونده‌های فشرده‌ی جاوا (JAR) می‌پردازد. در ضمن، مفاهیم شیئگرا در رابطه با مثال‌های بخش ۱ و ۲ در آنجا ارائه شده‌اند.

برای کسب اطلاعات بیشتر در باره‌ی زبان برنامه‌نویسی جاوا از صفحه‌ی مقالات (*http://java.sun.com/developer/technicalArticles*)، آموزش (*http://java.sun.com/developer/onlineTraining*)، و سایر اسناد اطلاعاتی (*http://java.sun.com/developer/infodocs/*) شرکت سان مایکروسیستمز نیز می‌توانید استفاده کنید.

مونیکا پاولان (*http://java.sun.com/people/monicap*) یکی از نویسندگان عضو تیم JDC است. او در زمینه‌ی گرافیک دوبعدی و سه‌بعدی، امنیت، و محصولات پایگاه داده‌ای سابقه‌ی برنامه‌نویسی دارد، و علاقه‌مند به کاوش فناوری‌های نوین است. *monica.pawlan@eng.sun.com*

متن برنامه‌ها

AppendIO.java

```
import java.awt.Color;
import java.awt.BorderLayout;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;

class AppendIO extends JFrame implements ActionListener {
   JLabel text;
   JButton button;
   JPanel panel;
   JTextField textField;
   private boolean _clickMeMode = true;

   AppendIO() { //Begin Constructor
     text = new JLabel("Text to save to file:");
     button = new JButton("Click Me");
     button.addActionListener(this);
     textField = new JTextField(30);
     panel = new JPanel();
     panel.setLayout(new BorderLayout());
     panel.setBackground(Color.white);
     getContentPane().add(panel);
     panel.add(BorderLayout.NORTH, text);
     panel.add(BorderLayout.CENTER, textField);
     panel.add(BorderLayout.SOUTH, button);
   } //End Constructor

   public void actionPerformed(ActionEvent event){
     Object source = event.getSource();
     if (source == button){
       String s = null;
       if (_clickMeMode){
         try {
      //Write to file
          String text = textField.getText();
          byte b[] = text.getBytes();
          String outputFileName = System.getProperty("user.home",
                                  File.separatorChar + "home" +
                                  File.separatorChar + "zelda") +
                                  File.separatorChar + "text.txt";
          File outputFile = new File(outputFileName);
          RandomAccessFile out = new RandomAccessFile(outputFile,
  "rw");
          out.seek(outputFile.length());
          out.write(b);

     //Write a new line (NL) to the file.
         out.writeByte('\n');
         out.close();

     //Read from file
```

```
            String inputFileName = System.getProperty("user.home",
                                   File.separatorChar + "home" +
                                   File.separatorChar + "zelda") +
                                   File.separatorChar + "text.txt";
            File inputFile = new File(inputFileName);
            FileInputStream in = new FileInputStream(inputFile);
            byte bt[] = new byte[(int)inputFile.length()];
            in.read(bt);
            s = new String(bt);
            in.close();
          } catch(java.io.IOException e) {
            System.out.println(e.toString());
          }
//Clear text field
         textField.setText("");
//Display text read from file
          text.setText("Text retrieved from file:");
          textField.setText(s);
          button.setText("Click Again");
          _clickMeMode = false;
        } else {
//Save text to file
          text.setText("Text to save to file:");
          textField.setText("");
          button.setText("Click Me");
          _clickMeMode = true;
        }
    }
 }//end action performed method

 public static void main(String[] args) {
    JFrame frame = new AppendIO();
    frame.setTitle("Example");
    WindowListener l = new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        System.exit(0);
      }
    };
    frame.addWindowListener(l);
    frame.pack();
    frame.setVisible(true);
  }
}
```

apppolfile

```
/* AUTOMATICALLY GENERATED ON Mon Mar 08 16:12:26 PST 1999*/
/* DO NOT EDIT */

grant {
  permission java.awt.AWTPermission "accessEventQueue";
  permission java.awt.AWTPermission "showWindowWithoutWarningBanner";
  permission java.util.PropertyPermission "user.home", "read";
  permission java.io.FilePermission "${user.home}/text.txt", "write";
  permission java.io.FilePermission "${user.home}/text2.txt", "read";
};
```

ApptoAppl.java

```
import java.awt.Color;
```

```
import java.awt.BorderLayout;
import java.awt.event.*;
import javax.swing.*;
import java.applet.Applet;

public class ApptoAppl extends Applet
      implements ActionListener {

   JLabel text;
   JButton button;
   JPanel panel;
   private boolean _clickMeMode = true;

   public void init(){
     setLayout(new BorderLayout(1, 2));
     setBackground(Color.white);

     text = new JLabel("I'm a Simple Program");
     button = new JButton("Click Me");
     button.addActionListener(this);
     add("Center", text);
     add("South", button);
   }

  public void start(){
     System.out.println("Applet starting.");
  }

  public void stop(){
     System.out.println("Applet stopping.");
  }

  public void destroy(){
     System.out.println("Destroy method called.");
  }

   public void actionPerformed(ActionEvent event){
        Object source = event.getSource();
        if (_clickMeMode) {
          text.setText("Button Clicked");
          button.setText("Click Again");
          _clickMeMode = false;
        } else {
          text.setText("I'm a Simple Program");
          button.setText("Click Me");
          _clickMeMode = true;
        }
   }
}
```

Dba.java

```
import java.awt.Color;
import java.awt.BorderLayout;
import java.awt.event.*;
import javax.swing.*;
import java.sql.*;
import java.net.*;
import java.util.*;
import java.io.*;
```

```
class Dba extends JFrame implements ActionListener {

   JLabel text, clicked;
   JButton button, clickButton;
   JPanel panel;
   JTextField textField;
   private boolean _clickMeMode = true;

   private Connection c;

   final static private String _driver =
  "oracle.jdbc.driver.OracleDriver";
   final static private String _url =
  "jdbc:oracle:thin:username/password@(description=(address_list=(addr
  ess=(protocol=tcp)(host=developer)(port=1521)))(source_route=yes)(co
  nnect_data=(sid=ansid)))";


   Dba(){ //Begin Constructor
     text = new JLabel("Text to save to database:");
     button = new JButton("Click Me");
     button.addActionListener(this);
     textField = new JTextField(20);
     panel = new JPanel();
     panel.setLayout(new BorderLayout());
     panel.setBackground(Color.white);
     getContentPane().add(panel);
     panel.add(BorderLayout.NORTH, text);
     panel.add(BorderLayout.CENTER, textField);
     panel.add(BorderLayout.SOUTH, button);
   } //End Constructor

   public void actionPerformed(ActionEvent event){
     try{
// Load the Driver
        Class.forName (_driver);
// Make Connection
        c = DriverManager.getConnection(_url);
      }
      catch (java.lang.ClassNotFoundException e){
       System.out.println("Cannot find driver class");
       System.exit(1);
      }catch (java.sql.SQLException e){
       System.out.println("Cannot get connection");
       System.exit(1);
      }

     Object source = event.getSource();
     if(source == button){
       if(_clickMeMode){
         JTextArea displayText = new JTextArea();
         try{
      //Code to write to database
          String theText = textField.getText();
          Statement stmt = c.createStatement();
          String updateString = "INSERT INTO dba VALUES ('" + theText
  + "')";
        int count = stmt.executeUpdate(updateString);
     //Code to read from database
```

```
          ResultSet results = stmt.executeQuery("SELECT TEXT FROM dba
  ");
          while(results.next()){
            String s = results.getString("TEXT");
            displayText.append(s + "\n");
          }
          stmt.close();
        }catch(java.sql.SQLException e){
         System.out.println("Cannot create SQL statement");
        }

       //Display text read from database
         text.setText("Text retrieved from database:");
         button.setText("Click Again");
         _clickMeMode = false;
//Display text read from database
       } else {
         text.setText("Text to save to database:");
         textField.setText("");
         button.setText("Click Me");
         _clickMeMode = true;
       }
    }
}

   public static void main(String[] args){
      Dba frame = new Dba();
      frame.setTitle("Example");
      WindowListener l = new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
          System.exit(0);
        }
      };
      frame.addWindowListener(l);
      frame.pack();
      frame.setVisible(true);
    }
}
```

DbaAppl.java

```
import java.awt.Color;
import java.awt.BorderLayout;
import java.awt.event.*;
import java.applet.Applet;
import javax.swing.*;
import java.sql.*;
import java.net.*;
import java.io.*;

public class DbaAppl extends Applet implements ActionListener {

   JLabel text, clicked;
   JButton button, clickButton;
   JTextField textField;
   private boolean _clickMeMode = true;
   private Connection c;

   final static private String _driver =
  "oracle.jdbc.driver.OracleDriver";
```

```
 final static private String _url =
"jdbc:oracle:thin:username/password@(description=(address_list=(addr
ess=(protocol=tcp)(host=developer)(port=1521)))(source_route=yes)(co
nnect_data=(sid=ansid)))";

 public void init(){
   setBackground(Color.white);
   text = new JLabel("Text to save to file:");
   clicked = new JLabel("Text retrieved from file:");
   button = new JButton("Click Me");
   button.addActionListener(this);
   clickButton = new JButton("Click Again");
   clickButton.addActionListener(this);
   textField = new JTextField(20);
   setLayout(new BorderLayout());
   setBackground(Color.white);
   add(BorderLayout.NORTH, text);
   add(BorderLayout.CENTER, textField);
   add(BorderLayout.SOUTH, button);
 }

 public void start(){
   System.out.println("Applet starting.");
 }

 public void stop(){
   System.out.println("Applet stopping.");
 }

 public void destroy(){
  System.out.println("Destroy method called.");
 }

 public void actionPerformed(ActionEvent event){
   try{
      Class.forName (_driver);
      c = DriverManager.getConnection(_url);
    }catch (java.lang.ClassNotFoundException e){
     System.out.println("Cannot find driver");
     System.exit(1);
    }catch (java.sql.SQLException e){
     System.out.println("Cannot get connection");
     System.exit(1);
    }

   Object source = event.getSource();
   if(source == button){
     if(_clickMeMode){
       JTextArea displayText = new JTextArea();
       try{
   //Write to database
         String theText = textField.getText();
         Statement stmt = c.createStatement();
         String updateString = "INSERT INTO dba VALUES ('" + theText
+ "')";
         int count = stmt.executeUpdate(updateString);
  //Read from database
         ResultSet results = stmt.executeQuery("SELECT TEXT FROM dba
");
         while(results.next()){
```

```
            String s = results.getString("TEXT");
            displayText.append(s + "\n");
          }
          stmt.close();
         }catch(java.sql.SQLException e){
         System.out.println("Cannot create SQL statement");
         System.exit(1);
         }

//Display text read from database
        text.setText("Text retrieved from file:");
        button.setText("Click Again");
        _clickMeMode = false;
//Display text read from database
      } else {
        text.setText("Text to save to file:");
        textField.setText("");
        button.setText("Click Me");
        _clickMeMode = true;
      }
    }
  }
}
```

DbaOdbAppl.java

```
import java.awt.Font;
import java.awt.Color;
import java.awt.BorderLayout;
import java.awt.event.*;
import java.applet.Applet;
import javax.swing.*;

import java.sql.*;
import java.net.*;
import java.io.*;

public class DbaOdbAppl extends Applet
      implements ActionListener {

   JLabel text, clicked;
   JButton button, clickButton;
   JTextField textField;
   private boolean _clickMeMode = true;
   private Connection c;
   final static private String _driver =
  "sun.jdbc.odbc.JdbcOdbcDriver";
   final static private String _user = "username";
   final static private String _pass = "password";
   final static private String _url = "jdbc:odbc:jdc";
   public void init(){
     text = new JLabel("Text to save to file:");
     clicked = new JLabel("Text retrieved from file:");
     button = new JButton("Click Me");
     button.addActionListener(this);
     clickButton = new JButton("Click Again");
     clickButton.addActionListener(this);
     textField = new JTextField(20);
     setLayout(new BorderLayout());
     setBackground(Color.white);
     add(BorderLayout.NORTH, text);
```

```
    add(BorderLayout.CENTER, textField);
    add(BorderLayout.SOUTH, button);
  }

  public void start(){
  }

 public void stop(){
    System.out.println("Applet stopping.");
 }

 public void destroy(){
    System.out.println("Destroy method called.");
 }

 public void actionPerformed(ActionEvent event){
   try{
     Class.forName (_driver);
     c = DriverManager.getConnection(_url, _user, _pass);
    }catch (Exception e){
     e.printStackTrace();
     System.exit(1);
    }

    Object source = event.getSource();
    if(source == button){
      if(_clickMeMode){
        JTextArea displayText = new JTextArea();
        try{
    //Write to database
          String theText = textField.getText();
          Statement stmt = c.createStatement();
          String updateString = "INSERT INTO dba VALUES ('" + theText
 + "')";
          int count = stmt.executeUpdate(updateString);
   //Read from database
          ResultSet results = stmt.executeQuery("SELECT TEXT FROM dba
 ");
          while(results.next()){
            String s = results.getString("TEXT");
            displayText.append(s + "\n");
          }
          stmt.close();
         }catch(java.sql.SQLException e){
         System.out.println("Cannot create SQL statement");
         System.exit(1);
         }

//Display text read from database
        text.setText("Text retrieved from file:");
        button.setText("Click Again");
        _clickMeMode = false;
//Display text read from database
      } else {
        text.setText("Text to save to file:");
        textField.setText("");
        button.setText("Click Me");
        _clickMeMode = true;
      }
    }
```

```
  }
}
```

dbapol

```
grant{
  permission java.awt.AWTPermission "accessEventQueue";
  permission java.awt.AWTPermission "showWindowWithoutWarningBanner";
  permission java.util.PropertyPermission "user.home", "read";
  permission java.io.FilePermission "${user.home}/text.txt", "write";
  permission java.io.FilePermission "${user.home}/text2.txt", "read";
};
```

DbaServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

import java.sql.*;
import java.net.*;
import java.io.*;

public class DbaServlet extends HttpServlet {

    private Connection c;
    final static private String _driver =
  "sun.jdbc.odbc.JdbcOdbcDriver";
    final static private String _user = "username";
    final static private String _pass = "password";
    final static private String _url = "jdbc:odbc:jdc";


    public void doPost(HttpServletRequest request,
       HttpServletResponse response)
       throws ServletException, IOException
   {
  response.setContentType("text/html");
  PrintWriter out = response.getWriter();
        out.println("<title>Example</title>" +
                    "<body bgcolor=FFFFFF>");

        out.println("<h2>Button Clicked</h2>");

        String DATA = request.getParameter("DATA");

        if(DATA != null){
    out.println("<STRONG>Text from form:</STRONG>");
                out.println(DATA);
        } else {
                out.println("No text entered.");
        }

//Establish database connection
        try{
          Class.forName (_driver);
          c = DriverManager.getConnection(_url, _user,_pass);
        }catch (java.sql.SQLException e){
          System.out.println("Cannot get connection");
          System.exit(1);
```

```
        }catch (java.lang.ClassNotFoundException e) {
          System.out.println("Driver class not found");
        }

        try{
//Code to write to database
          Statement stmt = c.createStatement();
          String updateString = "INSERT INTO dba " + "VALUES ('" +
  DATA + "')";
          int count = stmt.executeUpdate(updateString);

//Code to read from database
          ResultSet results = stmt.executeQuery("SELECT TEXT FROM dba
  ");
          while(results.next()){
            String s = results.getString("TEXT");
            out.println("<BR><STRONG>Text from database:</STRONG>");
            out.println(s);
        }
         stmt.close();
        }catch(java.sql.SQLException e){
         System.out.println("Cannot create SQL statement");
         System.exit(1);
        }

        out.println("<P>Return to <A HREF=../dbaHTML.html>Form</A>");
        out.close();
   }
}
```

ExampleProgram.java

```
//A Very Simple Example
class ExampleProgram {

   public static void main(String[] args){

        System.out.println("I'm a Simple Program");
   }
}
```

ExampServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ExampServlet extends HttpServlet {

    public void doPost(HttpServletRequest request,
       HttpServletResponse response)
       throws ServletException, IOException
   {
  response.setContentType("text/html");
  PrintWriter out = response.getWriter();
        out.println("<title>Example</title>" +
                    "<body bgcolor=FFFFFF>");

        out.println("<h2>Button Clicked</h2>");
```

```
        String DATA = request.getParameter("DATA");

        if(DATA != null){
                out.println(DATA);
        } else {
                out.println("No text entered.");
        }

        out.println("<P>Return to <A
  HREF=../simpleHTML.html>Form</A>");
        out.close();
}
}
```

FileIO.java

```
import java.awt.Color;
import java.awt.BorderLayout;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;

class FileIO extends JFrame implements ActionListener {
   JLabel text;
   JButton button;
   JPanel panel;
   JTextField textField;
   private boolean _clickMeMode = true;

   FileIO() { //Begin Constructor
     text = new JLabel("Text to save to file:");
     button = new JButton("Click Me");
     button.addActionListener(this);
     textField = new JTextField(30);
     panel = new JPanel();
     panel.setLayout(new BorderLayout());
     panel.setBackground(Color.white);
     getContentPane().add(panel);
     panel.add(BorderLayout.NORTH, text);
     panel.add(BorderLayout.CENTER, textField);
     panel.add(BorderLayout.SOUTH, button);
   } //End Constructor

   public void actionPerformed(ActionEvent event){
     Object source = event.getSource();
//The equals operator (==) is one of the few operators
//allowed on an object in the Java programming language
     if (source == button) {
       String s = null;
       //Write to file
       if (_clickMeMode){
         try {
          String text = textField.getText();
          byte b[] = text.getBytes();
          String outputFileName = System.getProperty("user.home",
                                  File.separatorChar + "home" +
                                  File.separatorChar + "zelda") +
                                  File.separatorChar + "text.txt";
          FileOutputStream out = new FileOutputStream(outputFileName);
```

```
            out.write(b);
            out.close();
           } catch(java.io.IOException e) {
             System.out.println("Cannot write to text.txt");
           }
       //Read from file
            try {
            String inputFileName = System.getProperty("user.home",
                           File.separatorChar + "home" +
                           File.separatorChar + "zelda") +
                           File.separatorChar + "text.txt";
      File inputFile = new File(inputFileName);
             FileInputStream in = new FileInputStream(inputFile);
             byte bt[] = new byte[(int)inputFile.length()];
             in.read(bt);
             s = new String(bt);
             in.close();
            } catch(java.io.IOException e) {
              System.out.println("Cannot read from text.txt");
            }
          //Clear text field
            textField.setText("");
          //Display text read from file
            text.setText("Text retrieved from file:");
            textField.setText(s);
            button.setText("Click Again");
            _clickMeMode = false;
          } else {
//Save text to file
            text.setText("Text to save to file:");
            textField.setText("");
            button.setText("Click Me");
            _clickMeMode = true;
          }
      }
  }

  public static void main(String[] args){
    FileIO frame = new FileIO();
    frame.setTitle("Example");
    WindowListener l = new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        System.exit(0);
      }
    };
    frame.addWindowListener(l);
    frame.pack();
    frame.setVisible(true);
  }
}
```

FileIOAppl.java

```
import java.awt.Color;
import java.awt.BorderLayout;
import java.awt.event.*;
import javax.swing.*;
import java.applet.Applet;
import java.io.*;
```

```
public class FileIOAppl extends JApplet implements ActionListener {
    JLabel text;
    JButton button;
    JPanel panel;
    JTextField textField;
    private boolean _clickMeMode = true;

    public void init(){
    getContentPane().setLayout(new BorderLayout(1, 2));
    getContentPane().setBackground(Color.white);
    text = new JLabel("Text to save to file:");
    button = new JButton("Click Me");
    button.addActionListener(this);
    textField = new JTextField(30);
    getContentPane().add(BorderLayout.NORTH, text);
    getContentPane().add(BorderLayout.CENTER, textField);
    getContentPane().add(BorderLayout.SOUTH, button);
    }
 public void start() {
    System.out.println("Applet starting.");
}

 public void stop() {
    System.out.println("Applet stopping.");
 }

 public void destroy() {
    System.out.println("Destroy method called.");
 }

 public void actionPerformed(ActionEvent event){
    Object source = event.getSource();
    if (source == button) {
    String s = null;
//Variable to display text read from file
    if (_clickMeMode) {
        try {
    //Code to write to file
        String text = textField.getText();
        String outputFileName = System.getProperty("user.home",
                                File.separatorChar + "home" +
                                File.separatorChar + "zelda") +
                                File.separatorChar + "text.txt";
            FileWriter out = new FileWriter(outputFileName);
            out.write(text);
            out.close();

    //Code to read from file
            String inputFileName = System.getProperty("user.home",
                                   File.separatorChar + "home" +
                                   File.separatorChar + "zelda") +
                                   File.separatorChar + "text.txt";
        File inputFile = new File(inputFileName);
            FileReader in = new FileReader(inputFile);
            char c[] = new char[(int)inputFile.length()];
            in.read(c);
            s = new String(c);
            in.close();
            } catch(java.io.IOException e) {
            System.out.println("Cannot access text.txt");
```

```
            }
//Clear text field
    textField.setText("");
//Display text read from file
        text.setText("Text retrieved from file:");
        textField.setText(s);
        button.setText("Click Again");
        _clickMeMode = false;
        } else {
//Save text to file
        text.setText("Text to save to file:");
        button.setText("Click Me");
        textField.setText("");
        _clickMeMode = true;
        }
    }
    }//end action performed method
}
```

FileIOError.java

```
import java.awt.Font;
import java.awt.Color;
import java.awt.BorderLayout;
import java.awt.event.*;
import javax.swing.*;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.File;

class FileIOError extends JFrame
        implements ActionListener {

    JLabel text;
    JButton button;
    JPanel panel;
    JTextField textField;
    private boolean _clickMeMode = true;

    FileIOError(){ //Begin Constructor
    text = new JLabel("Text to save to file:");
    button = new JButton("Click Me");
    button.addActionListener(this);
    textField = new JTextField(20);

    panel = new JPanel();
    panel.setLayout(new BorderLayout());
    panel.setBackground(Color.white);
    getContentPane().add(panel);
    panel.add("North", text);
    panel.add("Center", textField);
    panel.add("South", button);
    } //End Constructor

    public void actionPerformed(ActionEvent event){
    Object source = event.getSource();
    if(source == button){
        if(_clickMeMode){
        JLabel label = new JLabel();
```

```
//Write to file
        try{
        String text = textField.getText();
                byte b[] = text.getBytes();

                String outputFileName =
  System.getProperty("user.home", File.separatorChar + "home" +
  File.separatorChar + "monicap") + File.separatorChar + "text.txt";
                File outputFile = new File(outputFileName);
        FileOutputStream out = new FileOutputStream(outputFile);
        out.write(b);
        out.close();
        }catch(java.io.IOException e){
                System.out.println("Cannot write to text.txt");
        }

//Read from file
        try{
        String inputFileName = System.getProperty("user.home",
  File.separatorChar + "home" + File.separatorChar + "monicap") +
  File.separatorChar + "text.txt";
        File inputFile = new File(inputFileName);
                FileInputStream in = new FileInputStream(inputFile);
        byte bt[] = new byte[(int)inputFile.length()];
        int i;
        i = in.read(bt);
        String s = new String(bt);
                label.setText(s);
                in.close();
    }catch(java.io.IOException e){
        System.out.println("Cannot read from text.txt");
    }
        text.setText("Text retrieved from file:");
        button.setText("Click Again");
        _clickMeMode = false;
        } else {
        text.setText("Text to save to file:");
        textField.setText("");
        button.setText("Click Me");
        _clickMeMode = true;
        }
    }
    }

    public static void main(String[] args){
    FileIO frame = new FileIO();
    frame.setTitle("Example");
    WindowListener l = new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
        System.exit(0);
        }
    };

    frame.addWindowListener(l);
    frame.pack();
    frame.setVisible(true);
    }
}
```

FileIOServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class FileIOServlet extends HttpServlet {
    public void doPost(HttpServletRequest request,
                       HttpServletResponse response)
                       throws ServletException, IOException
   {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<body bgcolor=FFFFFF>");
        out.println("<h2>Button Clicked</h2>");
        String data = request.getParameter("data");
        if (data != null && data.length() > 0) {
          out.println("<STRONG>Text from form:</STRONG>");
          out.println(data);
        } else {
          out.println("No text entered.");
        }
        try {
//Code to write to file
           String outputFileName=
           System.getProperty("user.home",
           File.separatorChar + "home" +
           File.separatorChar + "monicap") +
           File.separatorChar + "text.txt";
           FileWriter fout = new FileWriter(outputFileName);
           fout.write(data);
           fout.close();
//Code to read from file
           String inputFileName =
              System.getProperty("user.home",
              File.separatorChar + "home" +
              File.separatorChar + "monicap") +
              File.separatorChar + "text.txt";
           FileReader fin = new FileReader(inputFileName);
           char c[] = new char[(char)inputFileName.length()];
           fin.read(c);
           String s = new String(c);
           out.println("<P><STRONG>Text from file:</STRONG>");
           out.println(s);
           fin.close();
        } catch(java.io.IOException e) {
            System.out.println("Cannot access text.txt");
        }
        out.println("<P>Return to <A
  HREF=../simpleHTML.html>Form</A>");
        out.close();
    }
}
```

java.policy

```
grant {
  permission java.net.SocketPermission "*:1024-65535",
   "connect,accept,resolve";
  permission java.net.SocketPermission "*:80", "connect";
```

```
  permission java.awt.AWTPermission "accessEventQueue";
  permission java.awt.AWTPermission "showWindowWithoutWarningBanner";
  permission java.util.PropertyPermission "user.home", "read";
  permission java.io.FilePermission "${user.home}/text.txt", "write";
  permission java.io.FilePermission "${user.home}/text2.txt", "read";
};
```

LessonTwoA.java

```
class LessonTwoA {
   static String text = "I'm a Simple Program";
   public static void main(String[] args){
        System.out.println(text);
   }
}
```

LessonTwoB.java

```
class LessonTwoB {

   String text = "I'm a Simple Program";
   static String text2 = "I'm static text";

   String getText(){
        return text;
   }

   String getStaticText(){
  return text2;
   }

   public static void main(String[] args){
        LessonTwoB progInstance = new LessonTwoB();
        String retrievedText = progInstance.getText();
  String retrievedStaticText = progInstance.getStaticText();
        System.out.println(retrievedText);
  System.out.println(retrievedStaticText);
   }
}
```

LessonTwoC.java

```
class LessonTwoC {

   static String text = "I'm a Simple Program";

   static String getText(){
        return text;
   }

   public static void main(String[] args){
        String retrievedText = getText();
        System.out.println(retrievedText);
   }
}
```

LessonTwoD.java

```
class LessonTwoD {
```

```
   String text;

   LessonTwoD(){
     text = "I'm a Simple Program";
   }

   String getText(){
        return text;
   }

   public static void main(String[] args){
        LessonTwoD progInst = new LessonTwoD();
        String retrievedText = progInst.getText();
        System.out.println(retrievedText);
   }
}
```

polfile

```
/* AUTOMATICALLY GENERATED ON Mon Mar 08 13:33:59 PST 1999*/
/* DO NOT EDIT */

grant {
  permission java.util.PropertyPermission "user.home", "read";
  permission java.io.FilePermission "${user.home}/text.txt", "write";
  permission java.io.FilePermission "${user.home}/text2.txt", "read";
};
```

RemoteServer.java

```
import java.awt.Font;
import java.awt.Color;
import java.awt.BorderLayout;
import java.awt.event.*;
import javax.swing.*;

import java.io.*;
import java.net.*;

import java.rmi.*;
import java.rmi.server.*;

class RemoteServer extends UnicastRemoteObject
                   implements Send {

  private String text;

  public RemoteServer() throws RemoteException {
    super();
  }

  public void sendData(String gotText){
    text = gotText;
  }

  public String getData(){
    return text;
  }
```

```
  public static void main(String[] args){
    if(System.getSecurityManager() == null) {
      System.setSecurityManager(new RMISecurityManager());
    }

    String name = "//kq6py.eng.sun.com/Send";
    try {
      Send remoteServer = new RemoteServer();
      Naming.rebind(name, remoteServer);
      System.out.println("RemoteServer bound");
    } catch (java.rmi.RemoteException e) {
      System.out.println("Cannot create remote server object");
    } catch (java.net.MalformedURLException e) {
      System.out.println("Cannot look up server object");
    }
  }
}
```

RMIClient1.java

```
import java.awt.Color;
import java.awt.BorderLayout;
import java.awt.event.*;
import javax.swing.*;

import java.io.*;
import java.net.*;

import java.rmi.*;
import java.rmi.server.*;

class RMIClient1 extends JFrame
      implements ActionListener {

   JLabel text, clicked;
   JButton button;
   JPanel panel;
   JTextField textField;
   Socket socket = null;
   PrintWriter out = null;
   static Send send;

   RMIClient1(){ //Begin Constructor
     text = new JLabel("Text to send:");
     textField = new JTextField(20);
     button = new JButton("Click Me");
     button.addActionListener(this);

     panel = new JPanel();
     panel.setLayout(new BorderLayout());
     panel.setBackground(Color.white);
     getContentPane().add(panel);
     panel.add("North", text);
     panel.add("Center", textField);
     panel.add("South", button);
   } //End Constructor

  public void actionPerformed(ActionEvent event){
     Object source = event.getSource();
```

```
     if(source == button){
//Send data over socket
        String text = textField.getText();
        try{
          send.sendData(text);
   } catch (java.rmi.RemoteException e) {
     System.out.println("Cannot send data to server");
   }
        textField.setText(new String(""));
     }
  }

   public static void main(String[] args){
        RMIClient1 frame = new RMIClient1();
  frame.setTitle("Client One");
        WindowListener l = new WindowAdapter() {
                public void windowClosing(WindowEvent e) {
                        System.exit(0);
                }
        };

        frame.addWindowListener(l);
        frame.pack();
        frame.setVisible(true);

    if(System.getSecurityManager() == null) {
      System.setSecurityManager(new RMISecurityManager());
    }

    try {
      String name = "//" + args[0] + "/Send";
      send = ((Send) Naming.lookup(name));
    } catch (java.rmi.NotBoundException e) {
      System.out.println("Cannot look up remote server object");
    } catch(java.rmi.RemoteException e){
      System.out.println("Cannot look up remote server object");
    } catch(java.net.MalformedURLException e) {
      System.out.println("Cannot look up remote server object");
    }
  }
}
```

RMIClient2.java

```
import java.awt.Color;
import java.awt.BorderLayout;
import java.awt.event.*;
import javax.swing.*;

import java.io.*;
import java.net.*;

import java.rmi.*;
import java.rmi.server.*;

class RMIClient2 extends JFrame
      implements ActionListener {

   JLabel text, clicked;
   JButton button;
   JPanel panel;
```

```
JTextArea textArea;
Socket socket = null;
PrintWriter out = null;
static Send send;

RMIClient2(){ //Begin Constructor
  text = new JLabel("Text received:");
  textArea = new JTextArea();
  button = new JButton("Click Me");
  button.addActionListener(this);

  panel = new JPanel();
  panel.setLayout(new BorderLayout());
  panel.setBackground(Color.white);
  getContentPane().add(panel);
  panel.add("North", text);
  panel.add("Center", textArea);
  panel.add("South", button);
} //End Constructor

public void actionPerformed(ActionEvent event){
   Object source = event.getSource();

   if(source == button){
      try{
        String text = send.getData();
  textArea.append(text);
} catch (java.rmi.RemoteException e) {
  System.out.println("Cannot access data in server");
}
   }
}

 public static void main(String[] args){
      RMIClient2 frame = new RMIClient2();
frame.setTitle("Client Two");
      WindowListener l = new WindowAdapter() {
              public void windowClosing(WindowEvent e) {
                      System.exit(0);
              }
      };

      frame.addWindowListener(l);
      frame.pack();
      frame.setVisible(true);

  if(System.getSecurityManager() == null) {
    System.setSecurityManager(new RMISecurityManager());
  }

  try {
    String name = "//" + args[0] + "/Send";
    send = ((Send) Naming.lookup(name));
  } catch (java.rmi.NotBoundException e) {
    System.out.println("Cannot access data in server");
  } catch(java.rmi.RemoteException e){
    System.out.println("Cannot access data in server");
  } catch(java.net.MalformedURLException e) {
    System.out.println("Cannot access data in server");
  }
```

```
  }
}
```

Send.java

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Send extends Remote {

  public void sendData(String text) throws RemoteException;
  public String getData() throws RemoteException;
}
```

SimpleApplet.java

```
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Color;

public class SimpleApplet extends Applet{

  String text = "I'm a simple applet";

  public void init() {
   text = "I'm a simple applet";
   setBackground(Color.cyan);
  }

  public void start() {
        System.out.println("starting...");
  }

  public void stop() {
        System.out.println("stopping...");
  }

  public void destroy() {
        System.out.println("preparing to unload...");
  }

   public void paint(Graphics g){
   System.out.println("Paint");
   g.setColor(Color.blue);
   g.drawRect(0, 0,
        getSize().width -1,
        getSize().height -1);
   g.setColor(Color.red);
   g.drawString(text, 15, 25);
    }
}
```

simpleHTML.html

```
<HTML>
<HEAD>
<TITLE>Example</TITLE>
</HEAD>
<BODY BGCOLOR="WHITE">
```

```
<TABLE BORDER="2" CELLPADDING="2">
<TR><TD WIDTH="275">

<H2>I'm a Simple Form</H2>

Enter some text and click the Submit button.<BR>
Clicking Submit invokes
<A HREF="./ExampServlet.java">ExampServlet.java</A>,<BR>
which returns an HTML page to the browser.

<FORM METHOD="POST" ACTION="/servlet/ExampServlet">

<INPUT TYPE="TEXT" NAME="DATA" SIZE=30>

<P>
<INPUT TYPE="SUBMIT" VALUE="Click Me">
<INPUT TYPE="RESET">
</FORM>

</TD></TR>
</TABLE>

</BODY>
</HTML>
```

SwingUI.java

```
import java.awt.Color;
import java.awt.BorderLayout;
import java.awt.event.*;
import javax.swing.*;

class SwingUI extends JFrame
      implements ActionListener {

   JLabel text, clicked;
   JButton button, clickButton;
   JPanel panel;
   private boolean _clickMeMode = true;

   SwingUI(){ //Begin Constructor
     text = new JLabel("I'm a Simple Program");
     button = new JButton("Click Me");
     button.addActionListener(this);

     panel = new JPanel();
     panel.setLayout(new BorderLayout());
     panel.setBackground(Color.white);
     getContentPane().add(panel);
     panel.add(BorderLayout.CENTER, text);
     panel.add(BorderLayout.SOUTH, button);
   } //End Constructor

   public void actionPerformed(ActionEvent event){
        Object source = event.getSource();
        if (_clickMeMode) {
          text.setText("Button Clicked");
          button.setText("Click Again");
          _clickMeMode = false;
        } else {
```

```
            text.setText("I'm a Simple Program");
            button.setText("Click Me");
            _clickMeMode = true;
          }
    }

    public static void main(String[] args){
      SwingUI frame = new SwingUI();
      frame.setTitle("Example");
      WindowListener l = new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
          System.exit(0);
        }
      };

      frame.addWindowListener(l);
      frame.pack();
      frame.setVisible(true);
    }

}
```