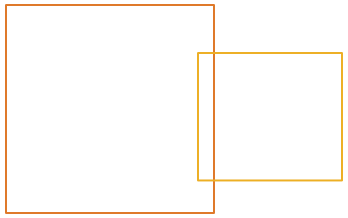
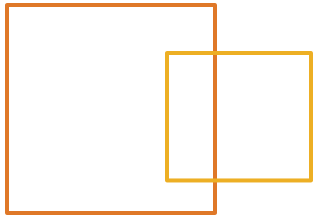
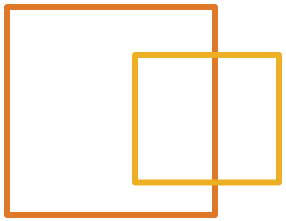
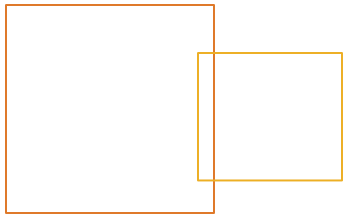
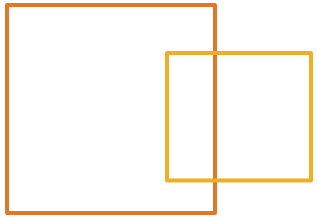
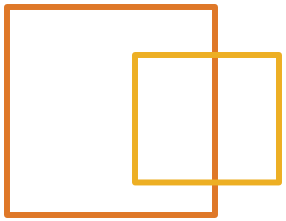





Fast Track to Java

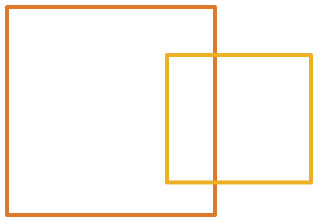
Customized for Starbucks

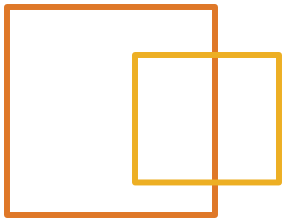
Delivered by DevelopIntelligence


Properties & Resource Bundles

Objectives





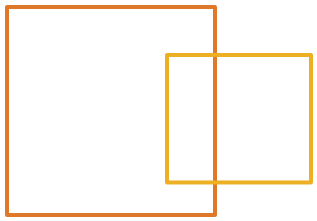
At the end of this module you should be able to:

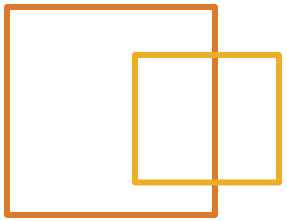
- Understand and use `Properties` to configure a program
- Understand how a `ResourceBundle` may be loaded using the system Locale, or a specific Locale, to configure a program's behavior according to the needs of a particular user
- Use `PropertyResourceBundle`
- Use `ListResourceBundle`

Environment, Arguments, Properties

- Java provides several mechanisms for moving configuration information from a command line into the program
  - Can use environment:
    - `Map<String, String> System.getenv()`
  - Can pass command line arguments
    - `public static void main(String [] args)`
  - Or can use properties:
    - `java –Dsome.property.value=1234`
    - `String s = System.getProperty(“some.property.value”);`

Properties





- Properties are usually defined with a dotted notation
  - The dots are just characters, no special hierarchy is created
- On the command line, property definition is
  - Introduced with the –D flag
  - No space in the property definition: `-Dprop.name=abcd`
  - Positioned before the class name, anything after the class to run will become an argument
  - Multiple definitions may be provided

Reading Properties From Files

- Properties may be loaded en masse from a text file

```
Properties props = System.getProperties();
props.load(new FileReader("my.properties"));
```

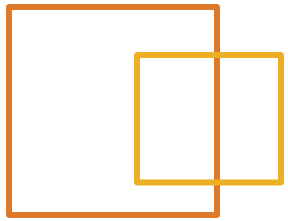
- Or from an XML file

```
props.loadFromXML(new FileInputStream("props.xml"));
```

- And may be written back out too:

```
props.store(
 new FileOutputStream("my.properties"),"Comments…");
props.storeToXML(
 new FileOutputStream("props.xml"), "Comments…");
```

Properties Formats



- Plain text properties are simply key=value pairs
  - Comments are introduced with '#'
- XML properties look like:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM
"http://java.sun.com/dtd/properties.dtd">
<properties>
  <comment>Here are some properties as XML</comment>
  <entry key="java.runtime.name">
    Java(TM) SE Runtime Environment
  </entry>
[...]
</properties>
```

ResourceBundle

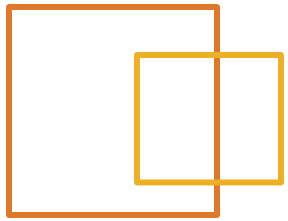
- `ResourceBundle` provides more flexibility than Properties
  - Locale-dependent configuration
  - Configuration of non-textual resources, e.g.
    - Windowing components
    - Program code for tax rules
- Resources are part of the program
  - They are loaded by the classloader and so may be:
    - Embedded in JAR files
    - Loaded over the net
  - Because of this, they are read-only

Locating a ResourceBundle

- Loader attempts to find the most specific match with the user's locale (ISO 639 and 3166):
  - Locale is language, region, [vendor specific variant]
    - en_US
    - fr_CA
- Resources are typically classes with a base-name, and then language and locale variations:
  - `HelpSystem_en_US.class`
  - `HelpSystem_en.class`
  - `HelpSystem.class`

```
ResourceBundle.getBundle(name, [locale])
```

PropertyResourceBundle



- `PropertyResourceBundle` is a subclass of `Resource` bundle
  - Unlike typical resource bundles, these are not classes, but are plain text property files
  - Provides for key, value pairs, just like properties
  - Less flexible—only supports text, but very often useful

PropertyResourceBundle Example

- Files located under `CLASSPATH`:
  - com/di/res/myresource.properties
  - com/di/res/myresource_en.properties
  - com/di/res/myresource_en_US.properties
  - And others (_en_GB, _fr_CA, _de_DE…)
- Each properties files contains key=value pairs specific to the language and conventions

PropertyResourceBundle Example

- To load a “suitable” bundle for user’s current locale:

```
ResourceBundle.getBundle(“com.di.res.myresource”);
```

- Can add explicit locale preferred over system default
  - Important with web-based clients of a server
- Notice the resource is described using a ***package***-like naming structure, ***not*** a directory-like one
  - This makes more sense with class-based resources
- If the system local is American English, this will search for these files, in order, and stop at the first (best) match:
  - myresource_en_US.properties
  - myresource_en.properties
  - myresource.properties

ListResourceBundle

- Base `ResourceBundle` class is abstract
- `ListResourceBundle` provides usual implementation for object-type resources
- Loading mechanism, selection of best-fit for locale, is same as `PropertyResourceBundle`
  - Note `ResourceBundle.getBundle` attempts to find classes first, then falls back to properties files
- Resources are created as classes/objects, and selected from the `ListResourceBundle` using a key
  - Still key/value pairs as before, but not restricted to text values; can use any object

ListResourceBundle Example

- Interfaces for objects to be looked up:

```
public interface IncomeTaxCalculator {
    public long calculateIncomeTax(long pay);
}
```

- Implementations for US and UK, e.g.:

```
public class UKIncomeTaxCalculator
    implements IncomeTaxCalculator {
    public long calculateIncomeTax(long pay) {
        return (long) (pay * 0.25);
    }
}
```

ListResourceBundle Example

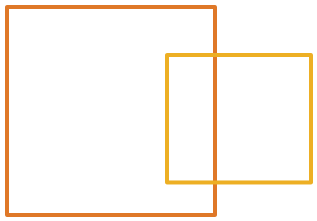
Create `ListResourceBundles` for _en_US and _en_GB

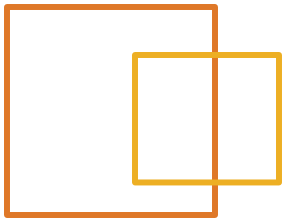
```
public class TaxesResourceBundle_en_US
  extends ListResourceBundle {
  protected Object[][] getContents() {
    return new Object[][]{
      {"location", "United States"},
      {"currency", "$"},
      {"income", new USIncomeTaxCalculator()},
      {"sales", new USSalesTaxCalculator()}
    };
  }
}
```

ListResourceBundle Example

```
public static void main(String[] args) {
  ResourceBundle rb =
    ResourceBundle.getBundle("tests.TaxesResourceBundle");
    // or could do: "tests.TaxesResourceBundle", Locale.UK

  System.out.println("Bundle location is “
    + rb.getObject("location"));

  System.out.println("Income tax on $100,000 is "
    + (rb.getObject("currency"))
    + ((IncomeTaxCalculator)(rb.getObject("income")))
      .calculateIncomeTax(100000));

  System.out.println("Sales tax on $100 is "
    + (rb.getObject("currency"))
    + ((SalesTaxCalculator)(rb.getObject("sales")))
      .calculateSalesTax(100));
```

Summary





In this module, we covered:

- Understand and use `Properties` to configure a program
- Understand how a `ResourceBundle` may be loaded using the system Locale, or a specific Locale, to configure a program's behavior according to the needs of a particular user
- Use `PropertyResourceBundle`
- Use `ListResourceBundle`