Outline

- Classes
  - Creating New Data Types: class
  - Methods, Arguments and Return Values

- Objects
  - You Manipulate Objects Using References
  - Primitives
  - You Never Destroy Objects

- Packages
  - Using Other Components
Creating New Data Types: class

• Class keyword defines new data type

  ```java
  class ATypeName { /* class body goes here */ }
  ATypeName a = new ATypeName();
  ```

• Fields

  ```java
  class DataOnly {
      int i;
      float f;
      boolean b;
  }
  ```

• Each instance of `DataOnly` gets its own copy of the fields

• In a class, primitives get default values
Methods, Arguments and Return Values

• Methods: how you get things done in an object
  – Traditionally called “functions”
  – Can only be defined inside classes

```java
ReturnType methodName(/* Argument list */) {
    // Method body
}
```

• Example method call:

```java
int x = a.f(); // For object a
```
You Manipulate Objects Using References

String s; // Reference only
// Normal object creation:
String s = new String("asdf");
// Special string initialization:
String s = "asdf";
Primitives

- Built-in types: *not* object references, but variables on the stack like C.
  - `boolean`, `char` (Unicode), `byte`, `short`, `int`, `long`, `float`, `double`
- Same operations as C/C++, same syntax
- Size of each data type is machine independent!
- Portability & performance implications
- To create objects, wrapper classes are provided:
  - `Boolean`, `Character`, `Byte`, `Short`, `Integer`, `Long`, `Float`, `Double`.

```java
char ch = 'x';
Character c = new Character(ch);
Or
Character c = new Character('x');
```
## Primitives Cont’d

<table>
<thead>
<tr>
<th>Primitive type</th>
<th>Size</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Wrapper type</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Boolean</td>
</tr>
<tr>
<td>char</td>
<td>16-bit</td>
<td>Unicode 0</td>
<td>Unicode $2^{16}-1$</td>
<td>Character</td>
</tr>
<tr>
<td>byte</td>
<td>8-bit</td>
<td>-128</td>
<td>+127</td>
<td>Byte</td>
</tr>
<tr>
<td>short</td>
<td>16-bit</td>
<td>$-2^{15}$</td>
<td>$+2^{15}-1$</td>
<td>Short</td>
</tr>
<tr>
<td>int</td>
<td>32-bit</td>
<td>$-2^{31}$</td>
<td>$+2^{31}-1$</td>
<td>Integer</td>
</tr>
<tr>
<td>long</td>
<td>64-bit</td>
<td>$-2^{63}$</td>
<td>$+2^{63}-1$</td>
<td>Long</td>
</tr>
<tr>
<td>float</td>
<td>32-bit</td>
<td>IEEE754</td>
<td>IEEE754</td>
<td>Float</td>
</tr>
<tr>
<td>double</td>
<td>64-bit</td>
<td>IEEE754</td>
<td>IEEE754</td>
<td>Double</td>
</tr>
<tr>
<td>void</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Void</td>
</tr>
</tbody>
</table>
You Never Destroy Objects

- **Scope of objects**

```java
{ // ←-- Beginning of scope
    String s = new String("a string");
} // ←-- End of scope
// Reference has gone "out of scope"
// but the object itself still exists
```
static Data ("class data")

- Normally each object gets its own data
- What if you want only one piece of data shared between all objects of a class?

class WithStaticData {
    static int x;
    int y;
}
WithStaticData
a = new WithStaticData(),
b = new WithStaticData(),
c = new WithStaticData();
Using Other Components

• Bring in a library of components using `import` keyword
• To specify particular element in library:
  – `import com.bruceecckel.utility.MyClass;`
• To specify entire library:
  – `import java.util.*;`
Package: the Library Unit

• Managing “name spaces”
  – Class members are already hidden inside class
  – Class names could clash
  – Need completely unique name even over the Internet

• Compilation units (.java files)
  – Name of .java file == name of single public class
  – Other non-public classes: not visible
  – Each class in file gets its own .class file
  – Program is a bunch of .class files (no .obj or .lib)
Creating a Library of Classes

- `package mypackage;`
  - `public` class is under the umbrella `mypackage`
  - Client programmer must `import mypackage.*;`

- Creating unique package names
  - Location on disk encoded into package name
  - Convention: first part of package name is Internet
  - domain name of class creator (reversed)
  - Java interpreter uses CLASSPATH environment
  - variable as starting point for search