Static revisited

Static methods

```
// Example:
// Java's built in Math class
public class Math {
    public static int abs(int a) {
        if (a >= 0) {
            return a;
        } else {
            return -a;
        }
    }
    public static double toDegrees(double radians) {
        return radians * 180 / PI;
    }
}

// Using the class:
System.out.println(Math.abs(-5));
//didn't need to create any object
```

Static methods

- **static**: Part of a class, not part of an object.
- Static methods:
  - Do not require an instance of the class and do not understand the *implicit parameter*, `this`; therefore, cannot access an object's instance variables
  - good for code related to a class but not to each object's state
  - if `public`, can be called from inside or outside the class

Static variables

- **static**: Part of a class, rather than part of an object.
  - Classes can have *static variables*.
  - Static variables are not replicated in each object; **a single variable is shared by all objects of that class.**

```
private static type name;
```

or,

```
private static type name = value;
```

  - Example:

```
private static int count = 0;
```

Example

- You are writing a class to represent a bank account, and you would like the constructor to automatically assign a running number as the account number.
- How can static variables help you?

Assigning ids for BankAccount

```
public class BankAccount {
    // static variable for assigning an account number
    // (shared among all instances of the class)
    private static int lastAssignedNumber = 1000;

    // instance variables(replicated for each object)
    private float balance;
    private int id;

    public BankAccount(float initial_balance) {
        lastAssignedNumber++;     // advance the id
        id = lastAssignedNumber; // give number to account
        balance = initial_balance;
    }
    ...
    public int getID() {    // return this account's id
        return id;
    }
}
```

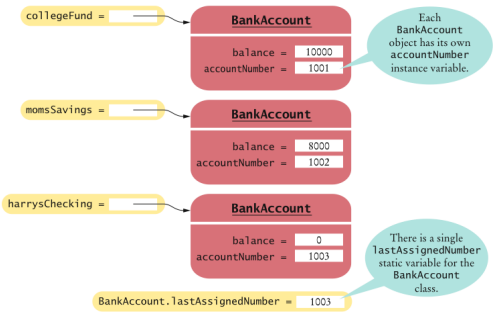

Figure from: Big Java by Cay Horstmann

Static variables

- Initializing static variables
  1. *Do nothing. variable is initialized with* `0` *(for numbers),* `false` *(for boolean values), or* `null` *(for objects)*
  2. *Use an explicit initializer, such as*

```
public class BankAccount
{
   ...
   private static int lastAssignedNumber = 1000;
   // Executed once
}
```

- Static variables should usually be declared `private`

Static variables

- Exception: Static constants, which may be either private or public:

```
public class BankAccount
{
   ...
   public static final double OVERDRAFT_FEE = 5;
   // Refer to it as BankAccount.OVERDRAFT_FEE
}
```

- Minimize the use of static variables (static final variables are ok)

Examples in the Java library

- Static variables in the `System` class:
  - `System.in` and `System.out`.
- And in the Java Math class:

```
public class Math {
   public static final double PI = 3.141592653589793;
   public static final double E = 2.718281828459045;

   ...
}
```

Java packages

Savitch Chapter 6.7

Creating a Java Package

Shape.java

```
// a shape stores its position
// on the screen
public abstract class Shape {
  int x,y;
  public Shape(int x, int y){
    this.x = x;
    this.y = y;
  }
}
```

Rectangle.java

```
public class Rectangle extends Shape
{
  double width, height;
  public Rectangle(int x, int y,
                   double h, double w ) {
    super(x, y);
    width = w;
    height = h;
  }
}
```

Circle.java

```
public class Circle extends Shape {
  double radius;
  public Circle(int x, int y, double r) {
    super(x, y);
    radius = r;
  }
}
```

Some motivation

- A few observations about the classes/ interfaces on the previous slide:
  - They are related, so it makes sense to group them together
  - Somebody else may have created a Shape or Rectangle class – name conflicts (e.g. with java.awt.Rectangle)
  - Classes within a package can be allowed to have unrestricted access to one another yet still restrict access outside the package.

Java packages

- Package: a named collection of related classes that are grouped in a directory (the name of the directory is the same as the name of the package).

Creating a Java Package

Shape.java

```
package shapes;
public abstract class Shape {
  int x,y;
  public Shape(int x, int y){
    this.x = x;
    this.y = y;
  }
}
```

Rectangle.java

```
package shapes;
public class Rectangle extends Shape
{
  double width, height;
  public Rectangle(int x, int y,
                   double h, double w ) {
    super(x, y);
    width = w;
    height = h;
  }
}
```

Circle.java

```
package shapes;
public class Circle extends Shape {
  double radius;
  public Circle(int x, int y, double r) {
    super(x, y);
    radius = r;
  }
}
```

Shape.java

```
package shapes;
public abstract class Shape {
  int x,y;
  public Shape(int x, int y){
    this.x = x;
    this.y = y;
  }
}
```

put the package statement in all the Java files you intend to include in your package.

needs to be the first statement in the file (except for comments)

- A package defines a *namespace*
- If you do not use a **package statement**, your class or interface ends up in the ***default package***, which is a package that has no name.

Using packages

- Only public package members are accessible outside the package in which they are defined. To use a public package member (class, interface) from outside its package, you must either:
  - Refer to the member by its long (disambiguated) name.
    - java.awt.Rectangle rectangle = new java.awt.Rectangle();
  - Import the member's entire package (not recommended).
    - import java.awt.*;

      Rectangle rectangle = new Rectangle();
  - Import the package member (recommended).
    - import java.awt.Rectangle;

      Rectangle rectangle = new Rectangle();

Package naming

- Package naming convention
  - The name is lower case so it isn't confused with a type or interface
  - All official Java packages start with java or javax.

Summary

- Packages:
  - Group together related Java types
  - Help avoid name conflicts
  - Provide access control
- For more information:

http://docs.oracle.com/javase/tutorial/java/package/index.html

Exceptions revisited

- Until now you only used predefined Java exceptions.
- You can write your own!
- Why would you want to do that?

Savitch Chapter 9

Example

```
public class DivideByZero Exception extends Exception {
   public DivideByZeroException() {
      super("Divide by zero");
   }
   public DivideByZeroException(String message) {
      super(message);
   }
}
```