Inheritance and Exceptions

Inheritance

Suppose you need to write a class X whose role would be very similar to an existing class Y. You could:

- Rewrite the whole code anew
  - Time consuming, introduces new bugs, makes maintenance a headache
- Copy the code of Y into X, then make your changes
  - Maintenance problem: you need to maintain both X and Y
- Inherit the code from Y, but override certain methods
  - Code common to X and Y is kept in Y. New methods are added in X

Inheritance - Example

You want to extend sportTeam to make it specific to certain sports:

- hockeyTeam
  - Has all the members defined in sportTeam, but also number of ties.
  - Number of points = 3 * victories + 1 * ties
- baseballTeam
  - Has all the members defined in sportTeam, but also number of homeruns

Types and dispatch

public class hockeyTeam extends sportTeam {
    private int ties;
    public hockeyTeam() { // constructor for hockeyTeam
        super(); // super() calls the constructor of the superclass
        ties=0;
    }
    public void addWin() {
        super.addWin(); /* This calls the addWin method provided by the parent class */
        points++; // Since points is private, this wouldn't compile. We need to declare points as "protected" instead of private to allow access to subclasses */
    }
    public void addTie() {
        ties++; points++;
    }
}

Read Little Book on Java, section 12-16!
Exceptions - When things go wrong

- Some things are outside programmer's control:
  - User types "Go Expos" when asked to enter number of victories
  - Try to open a file that doesn't exist
  - Try to compute sqrt(-1)
  - ...
- Exception mechanism allows to deal with these situations gracefully
  - When problem is detected, the code throws an exception
  - The execution of the program stops.
  - JVM looks for somebody to catch the exception
  - The code that catches the exception handles the problem, and execution continues from there
  - If no code catches exception, the program stops with error message
- An exception is an object that contains information about what went wrong.

Throwing exceptions

Syntax:
```
try {
    <block of code>
}catch (exceptiontype1 e1) {
    <block of code>
}catch (exceptiontype2 e2) {
    <block of code>
}...
finally {
    <block of code>
}
```

```
static double mySqrt(double x) {
    try {
        if (x<=0)
            throw new ArithmeticException("Sqrt is defined only for positive numbers");
        /* Code for computing sqrt goes here */
        try {
            x = mySqrt(10);
            y = mySqrt(-2);
            z = mySqrt(100);
        } catch (ArithmeticException e) {
            System.out.println("The mySqrt operation failed with error: "+ e);
            return 0;
        }
    } finally {
        <block of code>
    }
}
```

Methods throwing exceptions

- Sometimes, it is not appropriate for a method to handle the exception it threw
- Methods can throw exceptions back to the caller:

```
static double mySqrt(double x)
    throws ArithmeticException {
    if (x<0) {
        throw new ArithmeticException("Sqrt of "+ x + " is not defined");
    }
    /* Code for computing sqrt goes here */
}
```

```
public static void main(String args[]) {
    double x = 0, y = 0, z = 0;
    try {
        x = mySqrt(10);
        y = mySqrt(-2);
        z = mySqrt(100);
    } catch (ArithmeticException e) {
        System.out.println(e.toString());
    } // what is the value of x, y, z now?
    // x is 1, y and z are zero
}
```

Java resources

- Java Application Programming Interface (API)
  [http://java.sun.com/j2se/1.4.2/docs/api/](http://java.sun.com/j2se/1.4.2/docs/api/)
- Java tutorials:
  [http://java.sun.com/j2se/1.4.2/docs/api/](http://java.sun.com/j2se/1.4.2/docs/api/)
- Java books:
  1594 different books on Amazon
  - Java: The Programming Language (3rd Edition) -- by Ken Arnold (Author), et al;
    By the authors of Java itself. The ultimate reference. Not easy to read for beginners.
    A text version of the Java API