COMP 202 – Week 6

- **We’ve seen how to write basic classes and create objects. Next, we will examine particular relationships between objects, how to associate variables and methods with the class itself (rather than with an object of the class), and revisit methods in more detail.**
- **This week we focus on:**
  - **Object relationships: aggregation (the "has-a" relation)**
  - **Static members (variables & methods)**
  - **Method decomposition**
  - **Method overloading**

An aggregate Bus object

- **An *aggregate object* is an object that contains references to other objects**
- **An aggregate object represents a *has-a relationship.***
- **A bus has a maximum speed, a number of seats, and a number of passengers.**
- **A bus also *has a* driver.**
- **A bus can travel, pick up passengers, drop off passengers, and change drivers.**
- **see Bus.java**
- **A driver has a name, an age, and a risk-taking factor.**
- **A driver can accept or refuse a passenger, and may fall asleep at the wheel.**
- **see Driver.java, QuebecBusTour.java**

Writing Classes

- Sometimes an object has to interact with other objects of the same type
- For example, we might add two `Rational` number objects together as follows:

```
r3 = r1.add(r2);
```

- One object (`r1`) is executing the method and another (`r2`) is passed as a parameter

- See RationalNumbers.java
- See Rational.java

The static Modifier

- The `static` modifier can be applied to variables as well as methods

- It associates a variable or method with the class rather than an object

- To make a variable or method static, we apply the `static` modifier to the declaration

- Invoked through the class name rather than through a particular object

Class Variables

- Static variables are sometimes called *class variables*
- Normally, each object has its own data space
- If a variable is declared as static, only one copy of the variable exists, ex:

```
private static float amount;
public static final double PI = 3.14159;
public static int numCircles;
```

- Memory space for a static variable is created as soon as the class in which it is declared is loaded
- Can be used always, even if class is never instantiated.
- To access class variables, use the class name.

Class Variables

- All objects created from the class share access to the static variable
- Changing the value of a static variable in one object changes it for all others
- Static methods and variables often work together
- See TheProudMotherCat.java
- See Cat.java

Class Methods

```
class SphereArea

public static final float PI=3.14159;

public static float area(float rad);
{
  return( 4 * PI * rad * rad );
}

class SphereTest

public static void main(String args[])
{
  Scanner scan = new Scanner(System.in);
  System.out.println("Enter radius of sphere: ");
  float radius = scan.nextFloat();
  float area = SphereArea.area(radius);
  System.out.println("Area of sphere is " + area);
}
```

Class Methods

```
public class SphereArea
{
  static final float PI=3.14159;

  public static void main(String args[])
  {
      Scanner scan = new Scanner(System.in);
      System.out.println("Enter radius of sphere: ");
      float radius = scan.nextFloat();
      System.out.println("Area of sphere is" +
                               area(radius) );
  }

  public static float area(float rad);
  {
      return( 4 * PI * rad * rad );
  }
}
```

Class Methods

- The order of the modifiers can be interchanged, but by convention visibility modifiers come first
- Recall that the `main` method is static; it is invoked by the system without creating an object
- Static methods cannot reference instance variables, because instance variables don't exist until an object exists
- However, they can reference static variables or local variables

Class Methods

- Equivalent to "global" methods, but no danger of name conflicts.
- Very similar to Class variables.
- `Math` and `System` are classes that only defines class methods because there is no appropriate object framework.
- Example: `Math.sqrt(i);`
- `Math` is the name of the class, and `sqrt()` is the name of class method (or static method).
- Example comparing the two types:
- see ComparingCats.java

Java Methods Revisited

- Reduce complexity
  - Abstraction
  - hide information so that you won't need to think about it
  - Minimize code size
- Improve maintainability and correctness
- Avoid duplicate code
- Limit effect of changes
- Hide the order in which events happen to be processed
- Improve performance
- Promote code reuse
- Make a section of code readable (short functions valuable)

Method Decomposition

- A method should be relatively small, so that it can be readily understood as a single entity
- A potentially large method should be decomposed into several smaller methods as needed for clarity
- Therefore, a service method of an object may call one or more support methods to accomplish its goal
- See PigLatin.java
- See PigLatinTranslator.java

Overloading Methods

- *Method overloading* is the process of using the same method name for multiple methods
- The *signature* of each overloaded method must be unique
- The signature includes the number, type, and order of the parameters
- The compiler must be able to determine which version of the method is being invoked by analyzing the parameters
- The return type of the method is <u>not</u> part of the signature

Overloading Methods

Version 1

```
float tryMe (int x)
{
   return x + .375;
}
```

Version 2

```
float tryMe (int x, float y)
{
   return x*y;
}
```

Invocation

```
result = tryMe (25, 4.32)
```

Overloaded Methods

- The `println` method is overloaded:

```
println (String s)
println (int i)
println (double d)
```

etc.

- The following lines invoke different versions of the `println` method:

```
System.out.println ("The total is:");
System.out.println (total);
```

Overloading Methods

- Constructors can be overloaded
- An overloaded constructor provides multiple ways to set up a new object

- See SnakeEyes.java
- See Die.java

- See SimpleCat.java