1.00 Lecture 10

Static Methods and Data

Reading for next time: Big Java: sections 7.1-7.4, 7.6, 7.8

Strings

```
public class StringExample {
  public static void main(String[] args) {
    String s= new String("Test");    // Strings are objects
    String first= "George ";         // Shortcut constructor
    String middle= "H.W. ";
    String last= "Bush";
    String full= first + middle + last;
    System.out.println("Full: " + full);

    // Testing for equality in strings (objects in general)
    String full2= "George H.W. Bush";
    if (full.equals(full2))          // Right way
        System.out.println("Strings equal");
    if (full == full2)               // Wrong way
        System.out.println("A miracle!");
    if (first == "George ")  // Wrong way,but sometimes works
        System.out.println("Not a miracle!");  // Unreliable
    // Modifying strings must be done indirectly-strings are constants
    middle= middle.substring(2, 4) + " ";  // Get 2nd, 3rd chars
    full= first + middle + last;
    System.out.println("Modified full: " + full);    }  }
    // See String documentation on javadoc
```

Static Class Methods, Data

- Static data fields:
  - Only one instance of data item for entire class
    - Not one per object
  - "Static" is a historic keyword from C and C++
  - "Class data fields" is a better term
    - These are the alternative to "instance data fields" (which are a field in each object)
- Static methods:
  - Do not operate on objects and do not use any specific object
  - Have access only to static data fields of class
    - Cannot access instance fields in objects
    - You can pass arguments to static methods, as with all methods
  - "Class methods" is a better term
    - These are the alternative to "instance methods" (that operate on an object)

When to Use Static Data

- Variables of which there is only one for a class
  - For example, the next ID number available for all MIT students (assuming they are issued sequentially). In a `Student` class:

```
public class Student {
  private String name;        // 1 value per instance
  private int ID;             // 1 value per instance
  private static int nextID=1; // 1 value per class
  public static int getID() { return nextID++;}
…
```

- Constants used by a class (`final` keyword)
  - Have one per class; don't need one in each object

```
  public static final int MAX_TERMS_AS_STUDENT= 16;
  public static final double ABSOLUTE_ZERO= 273.0;
```

  - If `ABSOLUTE_ZERO` is in class Temperature, it is invoked by

```
  double tKelvin= Temperature.ABSOLUTE_ZERO + tCelsius;
```

  - Constants are all caps by tradition (C, C++)
  - Static variables in C, C++ are different than in Java

When to Use Static Methods

- For methods that use only their arguments and thus don't need an object for member data

```
public static double pow(double b, double p)
// Math library, takes b to the p power
```

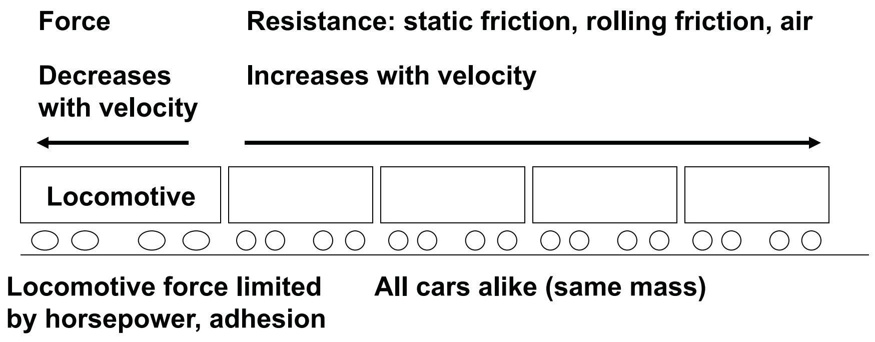
- For methods that only need static data fields

```
public static int getID( ) { return nextID++;}
// nextID is a static variable (see previous page)
```

- Main method in the class that starts the program
  - No objects exist yet for it to operate on
- All methods in C are like static Java methods, since C has no classes/objects
  - C++ has both Java-like and C-like methods

Exercise

- We'll experiment with whether rail locomotives have enough power to haul a train at a given velocity


Force
Resistance: static friction, rolling friction, air
Decreases with velocity
Increases with velocity
Locomotive
Locomotive force limited by horsepower, adhesion
All cars alike (same mass)

Exercise

- **Declare a class Train (Eclipse: File->New->Class)**
  - Create one public constant: gravity g= 9.8
  - You'll finish this class later
- **Declare a class Engine (Eclipse: File->New->Class)**
  - Variables: (there can be many engines w/diff mass, power)
    - Mass
    - Power
    - Coefficient of friction mu (0.3), a public constant for all engines
  - Constructor, as usual. *How many arguments does it have?*
  - getMass() method
  - getForce() method with one argument, velocity
    - f1= power/velocity (limit of engine horsepower)
    - f2= mass * g * mu (limit of adhesion to rail)
    - Return the minimum of f1, f2 (use Math.min() method)
- **Save / compile**

Exercise, p.2

- **Write a static version of getForce() in class Engine**
  - Supply all needed variables as arguments
  - Used by other classes that don't want to create an Engine object
  - Method overloading:
    - We can have multiple methods with the same name as long as they take a different number and or type of arguments.
    - We cannot have two methods that differ only in return type
    - Overloading is general; it's not related to static vs instance
- **To write this method:**
  - First, copy and paste the instance version of getForce() you just wrote
  - Then modify it into the static version
  - <u>You will need both versions of getForce() in this class</u>

Exercise, p.3

- Write class Car (Eclipse: File->New->Class)
  - Two private variables:
    - A single average mass for any car
    - Car type (coach, snack, first-class)
  - Constructor. *How many arguments does it have?*
  - Set and get methods for the single car mass
  - If you have time:
    - Write set and get methods for the car type
    - Are these instance or static methods?

Exercise, p. 4

- Finish class Train
- Data members:
  - Gravity g (already defined)
  - Constant c1= 0.00015 (rolling resistance)
  - Constant c2= 110.0 (air resistance)
  - One engine (object)
  - Number of cars (int)
  - (Which data members are static?)
- Constructor
  - What variables does it set?
- Method getNetForce(), with one argument: velocity
  - Compute weight= g*(engine mass + no of cars * car mass)
  - Compute net force= engine force - c1*weight*v - c2*v*v
  - Return net force

Exercise, p.5

- **Download TrainTest and add one line to it:**

```
public class TrainTest {
   public static void main(String[] args) {
        double vel= 30.0;      // 30 m/s, 70mph
        // Static method. No object needed.
        double f34= Engine.getForce(vel, 90000, 5500000);

        // Engine: 90 tonnes, 5500 kw
        Engine r34= new Engine(90000, 5500000);

        // Instance method
        double force34= r34.getForce(vel);

        // Don't need to create Cars. All we need is their mass
        // But we must set their mass:
        // Set it to 50000 kg here
        // Train
        Train amtrak41= new Train(r34, 10);
        // Instance method
        double force41= amtrak41.getNetForce(vel);
        // Output (run TrainTest)
   }
}
```