1.00 Lecture 10

Static Methods and Data

Reading for next time: Big Java: sections 13.1-13.7

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:

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

- 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
```

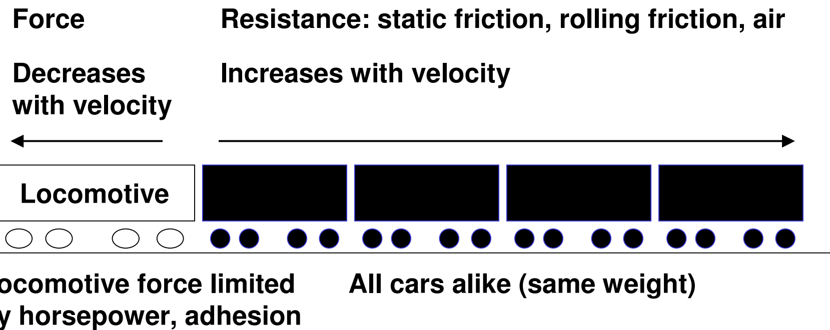
- For methods that only need static data fields

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



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
    - 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)
- 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 different arguments.
    - We cannot have two methods that differ only in return type
    - Overloading is general; it's not related to static vs instance

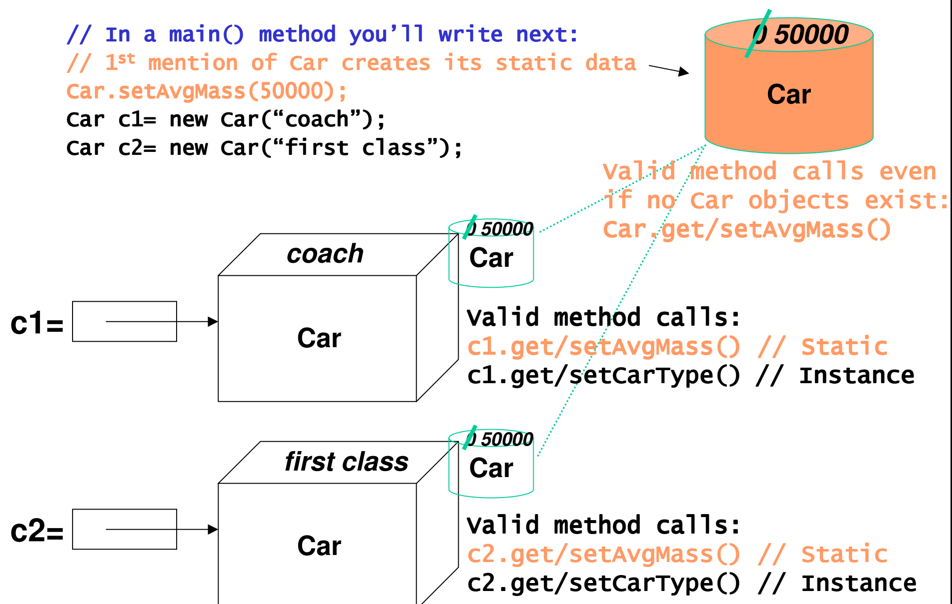
Exercise, p.3

- Write class Car (Eclipse: File->New->Class)
  - Two private variables:
    - A single mass for all cars
    - Car type (coach, snack, first-class)
  - Constructor. *How many arguments does it have?*
  - Set and get methods for the single car mass

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

Static data and methods



Exercise, p.5

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

```
public class TrainTest {
   public static void main(String[] args) {
         Engine r34= new Engine(90000, 5500000);    // 90 tonnes, 5500 kW
         double vel= 30.0;                          // 30 m/s, 70mph
         // Instance method
         double force34= r34.getForce(vel);
         // Static method
         double f34= Engine.getForce(vel, 90000, 5500000);

         // Don't need to create Cars. All we need is their mass
         // But we must set their mass: do it here

         // Train
         Train amtrak41= new Train(r34, 10);
         // Instance method
         double force41= amtrak41.getNetForce(vel);
         // Static method (if you had time)
         double f41= Train.getNetForce(vel, 10, r34);
   }
}
```

Solution: 2 engines, 2 trains

```
public class TrainTest3 {          // Solution with two trains, two engines
   public static void main(String[] args) {
         // Engines
         Engine r34= new Engine(90000, 5500000);    // 90 tonnes, 5500 kW
         Engine w96= new Engine(120000, 4000000);
         double vel= 30.0;                          // 30 m/s, 70mph
         // Instance methods
         double force34= r34.getForce(vel);
         double force96= w96.getForce(vel);
         // Static methods
         double f34= Engine.getForce(vel, 90000, 5500000);
         double f96= Engine.getForce(vel, 120000, 4000000);
         // Can't and don't need to create Cars, but set their avg wgt here
         Car.setAvgMass(50000);
         // Trains
         Train amtrak41= new Train(r34, 10);
         Train amtrak171= new Train(w96, 10);
         // Instance methods
         double force41= amtrak41.getNetForce(vel);
         double force171= amtrak171.getNetForce(vel);
         // Static methods
         double f41= Train.getNetForce(vel, 10, r34);
         double f171= Train.getNetForce(vel, 10, w96);
   }
}
```

Variable Lifecycles

- Instance (or object) variables
  - Created when their containing object is created
  - Initialized to default if not explicitly initialized
    - 0 for numbers, false for boolean, null for objects
  - Destroyed when Java garbage collector finds there are no remaining active references to object
- Static (or class) variables
  - Created when class is first used in program
  - Initialized to default if not explicitly initialized
    - 0 for numbers, false for boolean, null to objects
  - Usually exist for rest of program (unless unloaded)
- Local variables (or block variables)
  - Created in the statement where they're defined
  - <u>Not initialized by default. Contain unpredictable data</u>
  - Destroyed when block is exited (at ending brace)