Today

• Variables (mostly review)
• Constants
• Interfaces

Lab: Line following
Friday: interface implementation practice

Questions about Project 1?
Executing binary via Java?

- Use the `exec` methods from the `java.io.runtime` namespace.

Runtime rt = Runtime.getRuntime();
Process ps = rt.exec("executable.exe");
Mid-semestet feedback

1. What’s been most useful to you (and why)?

2. What could be going better / be more useful in class (and why)?

3. What could be going better / be more useful in lab (and why)?

4. What could students do to improve the class?

5. What could Matt do to improve the class?
Class Variables

• Also called instance variables
• Visible throughout class in which declared
• Should be private
  – Why?
• Provided with default value if not initialized (i.e., new ....)
Local Variables

• Visible only in methods in which declared
• May not be declared public or private
• Removed after method execution
• Must be given a value before use
Parameters

- Visible only in constructors and methods in which declared
- May **not** be declared `public` or `private`
- Associated with actual parameter during method invocation
- Removed after method execution
- Primitive types: **cannot change**
  
  (Object values: **can change**)
Guidelines

• Instance variables should always be private
  — Why?

• Method should be private if they are helpers -- otherwise public
  — Why?

• When should you make private helper methods?
public class Sphere {

    public static final double PI = 3.141592653589793; // pi is a universal constant

    public final double radius;
    public final double xpos;
    public final double ypos;
    public final double zpos;

    Sphere(double x, double y, double z, double r) {
        radius = r;
        xpos = x;
        ypos = y;
        zpos = z;
    }

    [...]
}
The Need for Flexibility

• Sometimes different classes are really very similar
  e.g. Circle and Rectangle
  • can be moved
  • draws something on the screen
  • could check if an \((x,y)\) point is inside the shape

• What if we want a variable that sometimes refers to a circle, and at other times a rectangle?
Interfaces

• Allow a programmer to exactly describe the features needed in a particular context

• Allow the use of parameters and variables that refer to values from more than one class

• The interface defines a contract and a class that implements the interface must follow the agreement
Movable

• An interface with two methods:

```java
public interface Movable {
    // Move receiver by dx in x direction and dy in y direction
    public void move(double dx, double dy);

    // Return whether receiver contains point
    public boolean contains(Location point);
}
```
Interfaces as Contracts

Movable geomObject;

– Implies we can call:
  • geomObject.move(5, 8);
  • geomObject.contains(pressPoint);

• Any object referred to by geomObject must have the methods move & contains
Interfaces & classes that implement them

• Interfaces
  – Provide *method headers*, not method bodies
  – May contain *constants*

• Classes implementing an interface
  – Must contain public methods *for each method* declared in the interface
  – Do not redefine constants in the interface
  – May implement multiple interfaces
public class FunnyFace implements Movable {
    private static final int FACE_HEIGHT = 60,
    FACE_WIDTH = 60,
    EYE_OFFSET = 20,
    EYE_RADIUS = 8,
    MOUTH_HEIGHT = 10,
    MOUTH_WIDTH = FACE_WIDTH/2;

    public void move( double dx, double dy ) {
        head.move( dx, dy );
        leftEye.move( dx, dy );
        rightEye.move( dx, dy );
        mouth .move( dx, dy );
    }

    public boolean contains( Location pt ) {
        
    }
}
public class StraightFace implements Movable {
    private static final int FACE_HEIGHT = 60,
        FACE_WIDTH = 60,
        EYE_OFFSET = 20,
        EYE_RADIUS = 0,
        MOUTH_HEIGHT = 0,
        MOUTH_WIDTH = FACE_WIDTH/2;
    
    public void move(double dx, double dy) {
        head.move(dx, dy);
        leftEye.move(dx, dy);
        rightEye.move(dx, dy);
        mouth.move(dx, dy);
    }
    
    public boolean contains(Location pt) {
        return false;
    }
}