Useless Fact of the Day

- The smallest independent nation in the world is Vatican City. It is only 0.2 square miles in size, and has a population of 921.

Creating a Variable

- Two steps:
  1. **Declare** the variable (if it's a local variable, declare it at the top of a method; if it's an instance variable, then declare it at the top of the class body):
     - `Coordinate myPoint;`
  2. **Initialize** (or **define**) the variable to some value (do this in a method, usually):
     - `myPoint = new Coordinate(3, 7);`
- Can optionally do both steps in one line:
  - `Coordinate myPoint = new Coordinate(3, 7);`

Topics

- Creating variables and constants
- Writing comments
- Odd Things:
  - Integer division, type casting, comparisons, primitive method arguments
Creating a Constant

• Both steps in one:
  • 1. Declare and initialize the constant on the same line (usually at the top of the class body):
  • `final int MY_CONSTANT = 7;`

Odd Thing #1: Integer Division

• What will “z” be if we initialize it like this:
  • `double z = 10 / 9;`
  • It will be “1.0” -- but why?
    • Integer division truncates
  • Integer division occurs only if both halves of the division are integers -- if either half is a double, then non-truncating double division will happen
  • `double z = 10.0 / 9;`
  • ...with the above line, z is initialized to “1.1111111111111112”

Odd Thing #2: Type Casting

• Methods expect to receive certain types of variables as arguments
• Sometimes, you will have to `cast` (convert) one variable to a different type in order to give it to a method
• You can convert between the number types (int, double, float), but not between, say, a double and a Coordinate
• You convert to a type by putting the desired type in parentheses before the variable/number to be converted: “(int) 5.7” will truncate the .7 and leave you with the integer 5
• As an example, GeneralPath’s “moveTo” method expects two `float` arguments. Let’s say our variable “g” is a GeneralPath. We can convert integers to floats:
  • `g.moveTo((float) 3, (float) 7);`

Odd Thing #3: Comparison

• You shouldn’t use `==` to compare Strings, or Sprites, or any other object
• It does not check to see if all of the contents of the objects are the same
• For Strings, we can use “s1.equals(s2)” to see if the String s1 is the same as the String s2
Odd Thing #4: Primitive Method Arguments

- When a method takes a primitive type of variable as an argument (int, double, char, boolean), it makes a local copy.
- If the method modifies the passed-in argument, it only modifies this copy, and not the original!
- The only way primitively-typed information can travel out of a method is with the method's return value.

**Primitive Method Arguments: Example**

```java
public int multiplyByThree(int z)
{
    z = z * 3;
    return z;
}
```

- Running “someMethod();” will output “2”:
  ```java
  public void someMethod()
  {
      double x = 2;
      multiplyByThree(x);
      System.out.println(x);
  }
  ```

- Running “someOtherMethod();” will output “6”:
  ```java
  public void someOtherMethod()
  {
      double x = 2;
      x = multiplyByThree(x);
      System.out.println(x);
  }
  ```

```
```