Random Numbers

- Java has (at least) 2 ways to generate random numbers
- There’s this way:
  ```java
  Random rand = new Random();
  int num = rand.nextInt(10);
  ```
- And this way:
  ```java
  int num = (int)(Math.random() * 10);
  ```
- What is the difference?

Random Class Object vs. Math Method

- First method is the way we originally learned to do things
  ```java
  Random rand = new Random();
  int num = rand.nextInt(10);
  ```
- Second method:
  ```java
  int num = (int)(Math.random() * 10);
  ```
- Where is the object to call the random() method?

Static Methods

- A static method in Java is one that can be called using the **name of a Class directly**
- The Math class has many such methods
  - **No need** for a Math object in most cases
  ```java
  public class Math {
      public static double pow(double b, double exp) {...}
      public static double random() {...}
      public static long round(double d) {...}
      public static double sqrt(double d) {...}
  }
  ```
The main() Method in Java Programs

- By convention, every Java program runs from a static main() method, so no object (such as Driver, that we’ve used) needs to be created
- Sometimes, however, it makes more sense to break our code up into separate methods, to allow easier organization and to remove repetition
- A problem: if there is no Driver object, then all these other methods will need to be static, too, or else we can’t call them, either!

```
private static DrawingGizmo pen;
public static void main(String[] args) {
    pen = new DrawingGizmo();
    // (1) draw a square
drawSquare();
    // (2) move over a bit
    pen.moveBy(50); // (3) draw another square
drawSquare();
}
```

Global Instance Variables

- So we can use our variable, pen, in both the main() and drawSquare() methods, we must declare it globally, as part of the overall class, not inside one particular method
- Now, it is available everywhere in the class definition
- Note: we must still instantiate it before it is ever used!

```
private static DrawingGizmo pen;
public static void main(String[] args) {
    pen = new DrawingGizmo();
    // (1) draw a square
drawSquare();
    // (2) move over a bit
    pen.moveBy(50); // (3) draw another square
drawSquare();
}
```

Static Methods and Variables

- When a method is static, it belongs to the class itself, rather than to individual instance objects
  - The class runs the method. It is not applied to an instance of the class.
  - Therefore, any other methods or instance variables (in this class) used by the static method must also be static, so everything can be done by the class itself
  - Thus, here both pen variable and drawSquare() method are static, too
Another, Better Solution

- Simply making everything in our program static is difficult since we often need methods that can be applied to instances.
- Sometimes, too, we will want multiple separate runs of a program each to do something different.
- Any static variable or method is shared by all class instances, and is always identical in every one of them since it can't access elements of different instances.
- One way to deal with this is to actually go ahead and create objects. We call a constructor to do so.
- Constructor has the same name as the class.
- The object, once created, can call other methods in that class or create more objects.
- Those objects/methods do not need to be static since we start with a dynamic object.

Review: `public` vs. `private` Method Access

- If we make a method in a class C private, it can only be used by object instances of the class C itself.
- If we make a method in C public, it can be used by:
  1. Any object instance from the class C itself.
  2. Any other class that runs the `C()` class constructor and creates an instance of an object of type C to use.
- When we write methods, we should decide if we want other coders to be able to use those methods or not.
  - **YES**: then make it public.
  - **NO**: then make it private.

Back to `main()`

- `main()` can therefore become very simple:
  ```java
  public static void main( String[] args )
  {
    Driver d = new Driver();
    d.drawSquare();
  }
  ```
- This will be the way we will write many of our programs from now on:
  - Keep `main()` quite simple.
  - Call one or more constructors to create some new objects.
  - Call some other methods on the objects if necessary, letting all complex coding go into those methods.

public vs. private Variable Access

- The same rules apply to instance (global) variables: private ones can only be `used` and `changed` by objects that belong to their same class, while public ones can be used and changed by **other classes**, too.
  - This can be very tricky to get right and can make code more fragile (badly engineered, especially when working as part of a group).

- If a global variable `window` in `Driver` were public, a user could write code like the following:
  ```java
  public class Main
  {
    public static void main( String[] args )
    {
      Driver d = new Driver();
      d.drawBlob( 0, 0 );
      d.drawBlob( 0, 400 );
      d.drawBlob( 400, 0 );
      d.drawBlob( 400, 400 );
    }
  }
  ```
  - Exception in thread "main" java.lang.NullPointerException
    at Driver.drawBlob(Driver.java:42)
    at Main.main(Main.java:10)

- If a global variable `window` in `Driver` were public, a user could write code like the following:
  ```java
  public class Main
  {
    public static void main( String[] args )
    {
      Driver d = new Driver();
      d.dontDraw();
    }
  }
  ```
  - Exception in thread "main" java.lang.NullPointerException
    at Driver.dontDraw(Driver.java:62)
    at Main.main(Main.java:4)

06 November 2019
Rule: Keep It private Unless Necessary

- When we write code, we can simplify our own lives, and those of other coders, by keeping everything we write as private as possible.
- If we make the things that we write private unless another class has to use them, then we can let the compiler catch code errors for us.

And remember: if we don’t say whether a method or instance variable is private or public, it will often be treated as if it is actually public automatically!

Access Settings ≠ Variable Scope!

- Remember: variables are always local to the scope {…} in which they are declared.
- Therefore, a variable declared inside a method is local to that method, and cannot be accessed outside the method, no matter if the method itself is private or public.
- Trying to make the variables private or public inside the method is just going to produce a compiler error.
- For methods and instance variables that are part of the class scope, private and public set access for code that is written outside the class.
- This does not affect methods in that same class.

This Week & Next

- Meetings this week:
  - Monday/Wednesday/Friday: regular classroom
  - Tuesday: in the CS Lab (16 Wing)
- Reading 06: Ch. 6 due Friday, Nov 8 at 4:00 PM
- Program 06: due 11:59 PM, Thursday November 7
- Midterm 2: Monday November 11
- Office Hours: Wing 212
  - Monday: 2:15 PM–2:45 PM
  - Tuesday: 2:30 PM–1:30 PM
  - Wednesday: 12:05 PM–1:00 PM
  - Friday: 2:15 PM–3:15 PM