Object-Oriented Programming: Using Classes

Reading: Appendix A

• Class basics and benefits.
• Creating objects using constructors.
• Calling Methods.
• Calling static methods and using static class variables.
• Using predefined Java classes.

Class Basics and Benefits:
Reading: Section A.2

• A class combines:
  • Data — identifiers that hold values. Can be any type (int, float, String, etc.)
  • Methods — code that manipulates the data.
• Classes are a template (or blueprint) used to create specific objects.
• All Java programs consist of at least one class.
• Two types of classes:
  • Application/Applet classes.
    • String and Scanner are examples.
  • Service classes.
    • The Math class.

• String name = "Patrick";
Class Basics and Benefits (continued):

- Example:
  - *BankAccount* class:
    - Data: name of account holder, account number, balance, mailing address, ...
    - Methods: set or get the value of each piece of data, compute new balance after a deposit or withdrawal
    - A specific instance of *BankAccount* might be an object called *myDreamAccount*.
      - Data: *Patrick; 024874898; $2,698,278.42; ...*

Class Basics and Benefits (continued):

Terminology:

- A class is declared **exactly** once in a program.
  - The declaration of a class can also be done in a library. The *Scanner* and *String* classes are examples.

- **Instances** of the class can be created. There can be many of these (analogous to having many *int*’s).

```java
String myName, yourName;
BankAccount yours, mine, ours, ourKids;
```

- **Object reference**: the identifier of the object.

  - The identifiers *myName, yourName, yours, mine, ours, ourKids* from above.

- **Instantiating an object**: creating an object of a class.

  ```java
  Scanner inputScan = new Scanner( System.in );
  String myName = "Patrick T. Homer";
  String myName = new String("Patrick T. Homer");
  String myInitials = new String("PTH");
  String myInitials = "PTH";
  ```
Class Basics and Benefits (continued):

**Terminology** (continued):

- **Instance of the class**: an object.
  ```java
  Scanner scanInput; // scanInput is an instance of Scanner class
  ```

- **Methods**: the code to manipulate the object data.
  ```java
  int xray = scanInput.nextInt(); // nextInt is a method of the Scanner class
  ```

- **Constructor**: special method that creates an object and assigns initial values to the data.
  ```java
  Scanner myInput;
  myInput = new Scanner( System.in ); // uses the constructor to create a Scanner object
  ```

- **Calling a method**: invoking the code to perform a service for an object.
  ```java
  String nextWord, anotherWord;
  anotherWord = new String("Bibble");
  shortWord = anotherWord.substring( 0, 3 ); // an invocation of the substring method
  nextWord = myInput.next(); // an invocation of the next method.
  ```

Class Basics and Benefits (continued):

**Encapsulation**:

- A class defines:
  - **Instance variables**: identifiers within the class that hold data values.
  - **Methods**: code that can manipulate (set, get, or change) the values of the identifiers.

- Each instance of a class (each object):
  - Has its own copy of the variables. The values of these variables are different for each instance.
    ```java
    String myName = "Patrick";
    String myWife = "Ann";
    ```
  - Different values of the variable inside each object.
  - Methods operate on the data.
    - Allow changes to the data.
    - Controls the value. For example,
      - A payroll object would not allow a negative hourly pay rate.
      - A baseball object would not allow the number of home runs to exceed the number of hits.
Class Basics and Benefits (continued):

Encapsulation (continued):

- The methods provide a protective shell around the data.
- Benefit:
  - The class methods ensure that the object data is always valid.
  - User (the programmer) cannot change the data directly.
    - Can only invoke the methods.
    - The methods limit how the data can be manipulated.

Class Basics and Benefits (continued):

Naming Conventions:

- Class names: start with an upper-case letter.
  - Capitalize internal words.
  - Examples:
    - Scanner
    - String
    - BankAccount
    - HomeAddress
- Object references: start with a lower-case letter.
  - Capitalize internal words.
  - Examples:
    - myName
    - scanInput
    - baseballPlayer
  - Same convention already discussed about identifier names.
**Calling Methods:**

**Object References:**

- Example:

  ```java
  String myName, myWife, ourDog, girlCat, boyCat;
  myName = new String("Patrick Homer");
  myWife = new String("Ann");
  ourDog = new String("Shishka");
  girlCat = new String("Shamatha");
  boyCat = new String("Thomas");
  ```

- The above code creates 5 `String` objects.
  - The identifier `myName` is a reference to the `String` object that contains “Patrick Homer”.
  - The identifier `myWife` is a reference to the `String` object that contains “Ann”.
  - The identifier `ourDog` is a reference to the `String` object that contains “Shishka”.
  - Etc.

**Calling Methods (continued):**

**Object References (continued):**

```java
myName = ourDog;
```

- The identifier `myName` is now a reference to the `String` object that contains “Shishka”.
  - Both `myName` and `ourDog` are references to the “Shishka” `String` object.
  ```java
  System.out.println("myName is " + myName);
  System.out.println("ourDog is " + ourDog);
  ```

- What happens to the `String` object “Patrick Homer”?
Class Scanner:

Reading: Section A.10, pages 662-664

- How do we (“us humans”) tell a program something?
  - Example: A program that will print my name:
    ```java
    String myName = "Patrick";
    System.out.println("The human’s name is " + myName);
    
    But, someone not named Patrick might want to use the program; hmmm…?
    
    - The Scanner class allows a program to read input from the keyboard.
    
    - Three steps:
      1. Tell Java that the Scanner class will be used:
         ```java
         import java.util.Scanner;
         ```
      2. Declare an instance of the Scanner class and connect it to the keyboard:
         ```java
         Scanner scanInput;
         scanInput = new Scanner( System.in );
         ```
      3. Read a value from the keyboard:
         ```java
         String myName = scanInput.next();
         ```

Class Scanner (continued):

- The first step tells Java that the program will use the Scanner class.
  - The Scanner class is part of a package of classes known as java.util
  - Put the import line before the declaration of your class:
    ```java
    import java.util.Scanner;
    public class ScannerSample
    { ...
    ```

Two ways to do this:

- To get just the one class:
  ```java
  import java.util.Scanner;
  ```
- To get to all classes in the java.util package:
  ```java
  import java.util.*;
  ```
Class Scanner (continued):

- The second step declares an instance of the Scanner class **inside** main:

```java
public static void main( String [] args )
{
    Scanner scanInput;
    scanInput = new Scanner( System.in );
```

- **Scanner** is a **class**, not a primitive type. We can declare instances of classes, just as we declare instances of primitive types (such as **int**, **float**, etc.).

- To give **scanInput** a value, we must create an instance of **Scanner**. The reserved word **new** is used for this.

- To have this instance of **Scanner** be connected to the keyboard, we use **System.in**.
  - Instances of **Scanner** can be connected to other “things”, such as files.
  - For now, we will be using only **System.in**.

```java
import java.util.Scanner;
public class ScannerSample
{
    public static void main(String[] args)
    {
        Scanner scanInput;
        scanInput = new Scanner( System.in );
```
Class Scanner (continued):

- The third step is to get the user’s input from the keyboard.
  - First, we ask the user:
    
    ```java
    System.out.print("Enter your name: ");
    ```
  - Note: use of `print`, not `println`. (What is the difference?)
  - Then, we get the answer:
    
    ```java
    String myName = scanInput.next();
    ```
  - Classes provide **methods** that are used to extract data values from the class.
  - The `next()` method of the `Scanner` class will read the next string the user types.
    - Declare a string, `myName`, to hold the answer returned by the `next()` method.

---

Class Scanner (continued):

- A complete program.
  
  ```java
  import java.util.Scanner;
  
  public class ScannerSample {
    public static void main(String[] args) {
      Scanner scanInput;
      scanInput = new Scanner(System.in);
      
      System.out.print("Enter your name: ");
      String myName = scanInput.next();
      
      System.out.println("myName is " + myName);
    } // end of main method
  } // end of class ScannerSample
  ```

- Questions:
  - What happens if the user types just one character?
  - What happens if the user types a really long name?
  - What happens if the user types characters other than letters?
  - What happens if the user types two (or more) names?
Class Scanner (continued):

- How does Scanner’s `next()` method “know” where the `String` ends?
  - When the user presses return or enter.
  - When a blank space is found.
  - When a tab is found.
  - Collectively, these (blank space, tab, newline) are known as *whitespace* characters.

Class Scanner (continued):

- One Scanner, many uses.

```java
import java.util.Scanner;

public class ScannerSampleAgain {
    public static void main(String[] args) {
        Scanner scanInput;
        scanInput = new Scanner(System.in);

        System.out.print("Enter your name: ");
        String myName = scanInput.next();

        System.out.println("myName is " + myName);

        System.out.print("Enter another name: ");
        myName = scanInput.next();

        System.out.println("myName is " + myName);
    } // end of main method
} // end of class ScannerSampleAgain
```
Class Scanner (continued):

- In general, methods within a class are used to:
  - **Get** the value of a data item contained in the class.
  - **Set** the value of a data item in the class.
  - **Perform** a calculation that, in part, involves data item(s) in the class.

- Methods that **get** the value of a data item have a **return type**.
  - The `next()` method of `Scanner` that was used on the previous slide, returns a `String`.
  - Other useful `Scanner` examples include:
    - `String nextLine()`
      - Returns a `String` that contains all the characters typed up to the next newline.
      - Provides a way to get blank spaces and tab characters included in the returned `String`.

```java
import java.util.Scanner;
public class ScannerInt {
    public static void main(String[] args) {
        Scanner scanInput;
        scanInput = new Scanner(System.in);
        System.out.print("Enter an integer: ");
        int myNumber = scanInput.nextInt();
        System.out.println("myNumber is "+myNumber);
    }
}
```

Class Scanner (continued):

- **int nextInt()**
  - Returns an `int` that contains the next integer typed on the keyboard.
    - Whitespace before the integer will be skipped.
  - What happens if:
    - The user types a number that contains a decimal point? a comma?
    - The user types several numbers on the same line?

- Example on how to scan an integer.
  ```java
  import java.util.Scanner;
  public class ScannerInt {
      public static void main(String[] args) {
          Scanner scanInput;
          scanInput = new Scanner(System.in);
          System.out.print("Enter an integer: ");
          int myNumber = scanInput.nextInt();
          System.out.println("myNumber is "+myNumber);
      }
  }
  ```
Class Scanner (continued):

- **double nextDouble()**
  - Returns a **double** that contains the next **double** typed on the keyboard.
  - Whitespace before the number will be skipped.
  - What happens if:
    - The user types a number that contains a decimal point? a comma?
    - The user types several numbers on the same line?

- Example on how to scan an **double**.

```java
import java.util.Scanner;

public class ScannerDouble {
    public static void main(String[] args) {
        Scanner scanInput;
        scanInput = new Scanner(System.in);
        System.out.print("Enter a double: ");
        double myNumber = scanInput.nextDouble();
        System.out.println("myNumber is " + myNumber);
    } // end of main method
} // end of class ScannerDouble
```

Class Scanner (continued):

- See Table A.17, page 662, for other useful **Scanner** methods:

<table>
<thead>
<tr>
<th>Return value</th>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>nextByte()</td>
<td>returns the next input as a byte</td>
</tr>
<tr>
<td>short</td>
<td>nextShort()</td>
<td>returns the next input as a short</td>
</tr>
<tr>
<td>int</td>
<td>nextInt()</td>
<td>returns the next input as an int</td>
</tr>
<tr>
<td>long</td>
<td>nextLong()</td>
<td>returns the next input as a long</td>
</tr>
<tr>
<td>float</td>
<td>nextFloat()</td>
<td>returns the next input as a float</td>
</tr>
<tr>
<td>double</td>
<td>nextDouble()</td>
<td>returns the next input as a double</td>
</tr>
<tr>
<td>boolean</td>
<td>nextBoolean()</td>
<td>returns the next input as a boolean</td>
</tr>
<tr>
<td>String</td>
<td>next()</td>
<td>returns the next token in the input line as a String</td>
</tr>
<tr>
<td>String</td>
<td>nextLine()</td>
<td>returns the input line as a String</td>
</tr>
</tbody>
</table>

- See also the Java API for **java.util.Scanner**
Calling Methods (continued):

Reading: Section A.5 and the Java API — http://docs.oracle.com/javase/7/docs/api/

- Objects contain data and methods.
  ```java
  myName = new String("Patrick Homer");
  ```
- The data is the name, *Patrick Homer* in this case.
- The methods can operate on the data and return information to us.

- Examples:
  - How long is my name?
    ```java
    int howLong;
    howLong = myName.length();
    System.out.println("There are " + howLong + " characters in my name");
    ```
  - Note: the empty ()’s are required since all methods have an argument list. For String’s `length` method, the argument list is empty.
  - Getting a character from the String:
    ```java
    char aLetter;
    aLetter = myName.charAt(4);
    ```
  - This time there is an argument. `charAt` takes one `int` argument.
  - The `charAt` method returns a `char`; thus, `aLetter` is of type `char`.

- The `substring` method returns a new `String` containing some of the characters from the original:
  ```java
  String myName, lastName;
  myName = new String("Patrick Homer");
  lastName = myName.substring(8, 13);
  System.out.println("Last name is " + lastName + ";
  ```
  - `lastName` is a reference to a `String` object that contains Patrick’s last name.
  - The method, `substring`, created a new `String` object.

- A variation:
  ```java
  int nameLength;
  nameLength = myName.length();
  lastName = myName.substring(8, nameLength);
  System.out.println("Last name is " + lastName + ");
  ```
- And another variation:
  ```java
  lastName = myName.substring(8, myName.length());
  System.out.println("Last name is " + lastName + ");
  ```
Calling Methods (continued):

- The `indexOf` method returns an `int` that is the index of the first occurrence of a character:

```java
public class IndexOfExampleOne
{
    public static void main(String[] args)
    {
        String myName = "Patrick Homer";
        String restOfName;
        int location;
        char aLetter = 'r';

        location = myName.indexOf( aLetter );
        System.out.println( aLetter + " first appears at position " + location);
        restOfName = myName.substring( location + 1, myName.length() );
        System.out.println("The rest of the name is " + restOfName + "]");
    } // end of main method
} // end of class IndexOfExampleOne
```

Calling Methods (continued):

- There are two versions of `indexOf`. The second version finds the first occurrence of a `String`:

```java
public class IndexOfExampleTwo
{
    public static void main(String[] args)
    {
        String myName = "Patrick Homer";
        String restOfName;
        int location;
        String lookFor = "rick";

        location = myName.indexOf( lookFor );
        System.out.println( lookFor + " starts at position " + location);
        restOfName = myName.substring( location, myName.length() );
        System.out.println("The rest of the name is " + restOfName + "]");
    } // end of main method
} // end of class IndexOfExampleTwo
```
Calling Methods (continued):

- The `toLowerCase` method returns a new `String` containing the original characters, but with all letters in lower case:
  ```java
  String myName, smallName;
  myName = new String("Patrick Homer");
  smallName = myName.toLowerCase();
  System.out.println("My name in lower case: "+ smallName + ");
  System.out.println("My name is still: "+ myName + ");
  ```

  - `smallName` is a reference to a `String` object that contains Patrick’s name in all lower case.
  - The method, `toLowerCase`, created a new `String` object.

- `toLowerCase` method is similar to `toUpperCase`, but returns the original string with all letters in upper case.
  ```java
  String myName, bigName;
  myName = new String("Patrick Homer");
  bigName = myName.toUpperCase();
  System.out.println("My name in upper case: "+ bigName + ");
  System.out.println("My name is still: "+ myName + ");
  ```

  - `bigName` is a reference to a `String` object that contains Patrick's name in all upper case.
  - The method, `toUpperCase`, created a new `String` object.
Calling Methods (continued):

• Summary: `String` methods covered; there are many more, see the Java API, [http://docs.oracle.com/javase/7/docs/api/](http://docs.oracle.com/javase/7/docs/api/):
  - `length()`
  - `charAt( int )`
  - `indexOf( char )`
  - `indexOf( String )`
  - `substring( int, int )`
  - `toLowerCase()`
  - `toUpperCase()`

Calling Methods (continued):

Reading: Java API: `java.text.DecimalFormat`

• The `DecimalFormat` class provides a way to control the appearance of numbers that are stored as `String`'s.
• You create an instance of `DecimalFormat` for each format you want to use.
• Note: Have to import the `DecimalFormat` class from the `java.text` package:
  ```java
  import java.text.DecimalFormat;  OR  import java.text.*;
  ```
• Example:
  • Want to print integers with the commas, as in: 27,387,982 or 37,298.
    ```java
    int oneNumber = 27387982;
    int nextNumber = 37298;
    DecimalFormat commaFormat;
    commaFormat = new DecimalFormat("#,##0");
    System.out.println("oneNumber = "+commaFormat.format(oneNumber));
    System.out.println("nextNumber = "+commaFormat.format(nextNumber));
    ```
  • The `#` symbol means to print a digit, but to not print leading zeroes.
  • The `,` in the format indicates where to put the commas in the final number.
Calling Methods (continued):

DecimalFormat (continued):

- The DecimalFormat class can specify floating-point (float and double) values as well as integer values.
- The format string: $$###.##$$ indicates the answer is to have no more than two decimal places.
- The value will be correctly rounded automatically when printed.

Example:

```java
import java.text.DecimalFormat;
public class FormatExample2
{
    public static void main(String[] args)
    {
        float oneNumber = 3.14159F;
        double nextNumber = 3827967.29836598263987649826395809384756;

        DecimalFormat commaFormat;
        commaFormat = new DecimalFormat("#,###.##");
        String myPi = commaFormat.format(oneNumber);

        System.out.println("oneNumber = " + commaFormat.format(oneNumber));
        System.out.println("nextNumber = " + commaFormat.format(nextNumber));
    } // end of main method
} // end of class FormatExample2;
```

Why only one decimal place?

Calling Methods (continued):

DecimalFormat (continued):

- If you need two (or more) different formats in a program, you can declare multiple instances of DecimalFormat.

```java
import java.text.DecimalFormat;
public class Pennies
{
    public static void main(String[] args)
    {
        float myBalance = 3426.07F;
        float myInterest;

        DecimalFormat dollarFormat = new DecimalFormat("$ #,##0.00");
        DecimalFormat pennyFormat = new DecimalFormat("#,###");

        myInterest = myBalance * 3.5F / 100;  // 3.5% interest rate

        System.out.print("Interest earned = ");
        System.out.println(dollarFormat.format(myInterest));

        System.out.print("Pennies earned = ");
        System.out.println(pennyFormat.format(myInterest * 100));
    } // end of method main
} // end of class Pennies
```
Calling Methods (continued):

Random (continued):

- Example using Random to generate dice rolls:

```java
import java.util.Random;

public class DiceRolls
{
    public static void main(String[] args)
    {
        Random dice = new Random();
        int dieOne, dieTwo, sum;
        dieOne = dice.nextInt(6) + 1;
        dieTwo = dice.nextInt(6) + 1;
        sum = dieOne + dieTwo;
        System.out.println(dieOne + " + " + dieTwo + " = " + sum);
    } // end of method main
} // end of class DiceRolls
```

This would not be correct. Why??

```java
int twoDice;
twoDice = dice.nextInt(11) + 2;
```
Calling Methods (continued):

**Random (continued):**

- To get values in a different range:
  ```java
  int start = 17;
  int end = 50;
  Random someValue = new Random();
  int answer = someValue.nextInt(end - start + 1) + start;
  System.out.println("answer = " + answer);
  ```
  - This will produce a random value in the range 17 to 50, inclusive.

  ```java
  int zap;
  // want zap in range 10..15
  zap = key.nextInt( 6 ) + 10;
  // want zap to be even
  zap = key.nextInt( 6 ) * 2 + 12;
  // want zap to be odd
  zap = (key.nextInt( 6 ) * 2 - 1) + 10;
  ```

---

**Unix Epoch**

Calling Methods (continued):

**Random (continued):**

- The `Random` class has a second constructor.
  - Takes one argument: a `long`.
    ```java
    Random zap = new Random( 142L );
    ```
  - Called a *seed*.
    - The mathematical formula for generating the pseudo-random numbers needs a starting value.
    - For a given starting value, the sequence of values generated will always be the same.
    - Change the seed to get a different value.
    - Useful in debugging a program that uses random numbers.
Calling Methods (continued):

Random (continued):

• Example using Random to generate dice rolls:

```java
import java.util.Random;

public class DiceRolls2
{
    public static void main(String[] args)
    {
        Random dice = new Random(17L);
        int dieOne, dieTwo;
        dieOne = dice.nextInt(6) + 1;
        dieTwo = dice.nextInt(6) + 1;
        System.out.println(dieOne + " + " + dieTwo + " = " + (dieOne + dieTwo));
        dieOne = dice.nextInt(6) + 1;
        dieTwo = dice.nextInt(6) + 1;
        System.out.println(dieOne + " + " + dieTwo + " = " + (dieOne + dieTwo));
    } // end of method main
} // end of class DiceRolls2
```

What happens if these ()'s are removed?

Static Classes: The Math Class:

Reading: section A.4, page 615, and the Java API: java.lang.Math

• The Math class is a static class.
  • Do not create an instance of Math.

• The Math class has two constants:

```java
System.out.println("pi is " + Math.PI);
pi is 3.141592653589793
System.out.println("e is " + Math.E);
e is 2.718281828459045
```

• Note the all caps: by convention in Java, constants are written in all caps.

• Example:

```java
public class Circle
{
    public static void main(String[] args)
    {
        float radius = 4.25F;
        System.out.println("Area = " + Math.PI * radius * radius);
        System.out.println("Circumference = " + 2 * Math.PI * radius);
    } // end of method main
} // end of class Circle
```
Static Classes: The **Math Class** (continued):

- **Absolute value, Math.abs():**
  - Can find the absolute value of any `int`, `long`, `float`, or `double`.
  - The return type will match the type of the argument:
    ```
    int xray = 4783;
    long zebra = -3928276268273986L;
    int answerOne = Math.abs(xray);   // returns an int since xray is an int
    long willow = Math.abs(xray);
    long answerTwo = Math.abs(zebra); // returns a long since zebra is a long
    int answerThree = Math.abs(zebra); // will not compile! Why??
    ```

- **Finding the minimum/maximum of two values, Math.min() and Math.max():**
  - The return type will match the type of the arguments:
    ```
    System.out.println("smaller is " + Math.min(278, 29876) );
    System.out.println("larger is " + Math.max(-2897.3897, -29837.83) );
    int xray, yoke = 17;
    xray = Math.min(yoke, yoke);
    int zebra = 17;
    xray = Math.max(yoke, zebra):
    ```

- **Raise to a power, Math.pow():**
  - Java does not have an exponentiation operator that would allow statements like: `answer = 2^5 + 2^6;`
  - The function `Math.pow()` provides this feature.
  - It takes two arguments, both of which are a `double`:
    - The number to be raised.
    - The power to use.
  - Example:
    ```
    double answer;
    answer = Math.pow(2, 5) + Math.pow(2, 6);
    System.out.println("The answer is " + answer);
    ```
Static Classes: The Math Class (continued):

• Raise to a power, Math.pow() (continued):
  
  • It can handle non-integer exponents; i.e., the following computes 613.75\(^{\frac{5}{2}}\)
    
    \[
    \text{answer} = \text{Math.pow}( 613.75, 2.5 );
    \]
    
    \[
    \text{System.out.println(}"\text{The answer is } + \text{answer});
    \]
    
    \[
    \text{answer} = \text{Math.sqrt( Math.pow( 613.75, 5.0 ) )};
    \]
    
    \[
    \text{System.out.println(}"\text{Gives the same answer: } + \text{answer});
    \]
  
  • Another Math.pow() example:
    
    \[
    \text{answer} = \text{Math.min( Math.pow(x, 3), Math.pow(y, 2) )};
    \]
    
    \[
    \text{answer} = \text{Math.pow( xray, 1./3. )};
    \]
  
  • One (but not the other) of these also computes 613.75\(^{\frac{5}{2}}\)

  • Which is correct, and which is wrong? Why is it wrong?
    
    \[
    \text{answer} = \text{Math.pow( 613.75, 7.0 / 2.0 )};
    \]
    
    \[
    \text{answer} = \text{Math.pow( 613.75, 5 / 2 )};
    \]
    
    \[
    \text{answer} = \text{Math.pow( 613.75, 5F / 2 )};
    \]

Static Classes: The Math Class (continued):

• Random double, Math.Random():
  
  • Does not take an argument.
  
  • Returns a double.
  
  • The value returned will be a random number in the range:
    
    • Greater than or equal to 0.
    
    • Less than 1.
  
  • Can convert the output to a range of integers by doing:
    
    \[
    \text{int answer, start = 8, end = 27;}
    \]
    
    \[
    \text{double aRandom;}
    \]
    
    \[
    \text{aRandom = Math.random();}
    \]
    
    \[
    \text{answer = start + (int) ( aRandom * ( end - start ) )};
    \]
    
    \[
    \text{System.out.println(}"\text{aRandom = } + \text{aRandom});
    \]
    
    \[
    \text{System.out.println(}"\text{answer = } + \text{answer});
    \]

In this course:

We will **not** use the Math.Random() method. Instead, we will use the java.util.Random class.