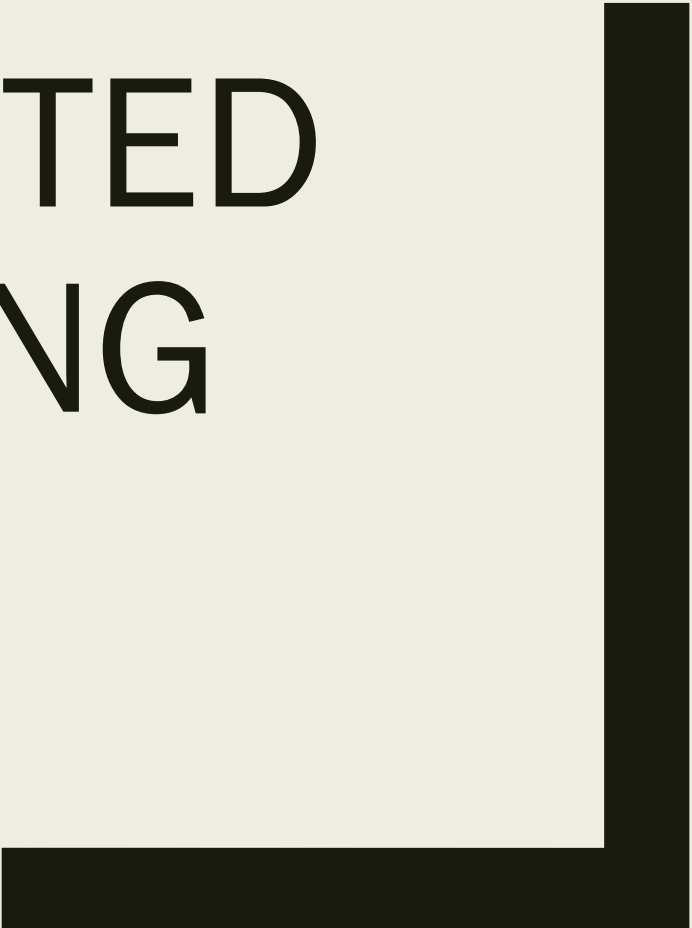




OBJECT ORIENTED PROGRAMMING

CMPINF 0401 Lab 06
Coin.java and CoinTester.java



Random Numbers

- Slide on the course website: <http://people.cs.pitt.edu/~hoffmant/java-slides/RandomNumbers.pdf>
 - *Look at example code*
- Seed
 - Consistent random numbers across all runs of a program
 - Great for debugging and grading
 - *No seed → Inconsistent random numbers across program runs*
- Modulus
 - *Think of modulus like the mathematical operation*
 - *If our modulus is 100, we cannot have a result of 100 since modulus only gives us remainder values (0 to 99). This is why it is an EXCLUSIVE upper bound*

How do declare, initialize, and use the Random object in Coin.java?

- We will declare our Random `r` object globally, but NOT initialize it up top. Since it is not a final, we will initialize it later on.
- Since our constructor accepts the seed value we need to initialize `r`, we can initialize it in the Coin constructor. Using the examples from the random slides, we know to set `r = new Random (seed)`
- Since we only need our `r` variable when flipping the coin, we will use it in our `flip()` method. We can make a temporary value called `int side` to hold our randomly generated number.
- To generate a random number that is 0 or 1, we can use 2 as our modulus since our remainder would have to be either 0 or 1. We can then compare this value to our finals HEADS and TAILS to determine if our flip was a heads or a tails

What methods do we need?

- First we find our constructor
 - *same object name as our file name (Coin → Coin.java)*
- Next look for any methods that use our new Coin object (coin1 or coin2)
 - *Ex: coin1.something() or coin2.something() means that something() is one of our Coin.java methods*
- Look carefully at what the tester wants the method to return (int, String, nothing)
- Look at the parenthesis to see if we need to declare any parameters in our method signature
 - *something() or something(value)*

```
1  /*
2      CoinTester.java - tests the Coin class by
3      constructing variables and calling it's methods
4      DO NOT MODIFY. DO NOT HAND IN. WE WILL USE THIS FILE TO TEST YOUR COIN CLASS
5  */
6  public class CoinTester
7  {
8      public static void main( String args[] )
9      {
10         Coin coin1 = new Coin( 17 ); // 17 is the SEED
11         Coin coin2 = new Coin( 13 ); // 13 is the SEED
12
13         // FLIP COIN1, PRINT RESULTS
14
15         System.out.println("\nFlipping Coin1 20 times.");
16         for (int i=0 ; i<20 ; ++i)
17             System.out.print( coin1.flip() + " " ); // Equal chance of head or tail
18         System.out.println();
19
20         System.out.println("# heads: " + coin1.getNumHeads() +
21                             " # tails: " + coin1.getNumTails() );
22         coin1.reset(); // sets numHeads and numTails back to zero;
23
24         // FLIP COIN2, PRINT RESULTS
25
26         System.out.println("\nFlipping Coin2 10 times.");
27         for (int i=0 ; i<10 ; ++i)
28             System.out.print( coin2.flip() + " " ); // Equal chance of head or tail
29         System.out.println();
30
31         System.out.println("# heads: " + coin2.getNumHeads() +
32                             " # tails: " + coin2.getNumTails() );
33         coin2.reset(); // sets numHeads and numTails back to zero;
34
35         // FLIP COIN1 AGAIN, PRINT RESULTS
36
37         System.out.println("\nFlipping Coin1 again 35 times.");
38         for (int i=0 ; i<35 ; ++i)
39             System.out.print( coin1.flip() + " " ); // Equal chance of head or tail
40         System.out.println();
41
42         System.out.println("# heads: " + coin1.getNumHeads() +
43                             " # tails: " + coin1.getNumTails() );
44         coin1.reset(); // calls private methods setNumHeads(0) and setNumTails(0)
45     } // END MAIN
46 } //END COINTESTER
```

What do we know about what we need?

- Coin Constructor
 - Parameters: *int seed*
 - Return Type: *NONE*
- `getNumHeads()`
 - Parameters: *none*
 - Return Type: *int*
- `getNumTails()`
 - Parameters: *none*
 - Return Type: *int*
- `flip()`
 - Parameters: *none*
 - Return Type: *String/char*
- `reset()`
 - Parameters: *none*
 - Return Type: *void*

```
1  /*
2     CoinTester.java - tests the Coin class by
3     constructing variables and calling it's methods
4     DO NOT MODIFY. DO NOT HAND IN. WE WILL USE THIS FILE TO TEST YOUR COIN CLASS
5  */
6  public class CoinTester
7  {
8      public static void main( String args[] )
9      {
10         Coin coin1 = new Coin( 17 ); // 17 is the SEED
11         Coin coin2 = new Coin( 13 ); // 13 is the SEED
12
13         // FLIP COIN1, PRINT RESULTS
14
15         System.out.println("\nFlipping Coin1 20 times.");
16         for (int i=0 ; i<20 ; ++i)
17             System.out.print( coin1.flip() + " " ); // Equal chance of head or tail
18         System.out.println();
19
20         System.out.println("# heads: " + coin1.getNumHeads() +
21                             " # tails: " + coin1.getNumTails() );
22         coin1.reset(); // sets numHeads and numTails back to zero;
23
24         // FLIP COIN2, PRINT RESULTS
25
26         System.out.println("\nFlipping Coin2 10 times.");
27         for (int i=0 ; i<10 ; ++i)
28             System.out.print( coin2.flip() + " " ); // Equal chance of head or tail
29         System.out.println();
30
31         System.out.println("# heads: " + coin2.getNumHeads() +
32                             " # tails: " + coin2.getNumTails() );
33         coin2.reset(); // sets numHeads and numTails back to zero;
34
35         // FLIP COIN1 AGAIN, PRINT RESULTS
36
37         System.out.println("\nFlipping Coin1 again 35 times.");
38         for (int i=0 ; i<35 ; ++i)
39             System.out.print( coin1.flip() + " " ); // Equal chance of head or tail
40         System.out.println();
41
42         System.out.println("# heads: " + coin1.getNumHeads() +
43                             " # tails: " + coin1.getNumTails() );
44         coin1.reset(); // calls private methods setNumHeads(0) and setNumTails(0)
45     } // END MAIN
46 } //END COINTESTER
```


Are there other methods in Coin that are not called in CoinTester?

- Yes! These methods will be **private** since they are only called internally by Coin.java
- The methods that were called in CoinTester.java will be **public** since they can be called externally or internally
- Getters and Setters – used to protect variables from being accessed or changed directly
 - *Getters (ex: getNumHead)*
 - Return (or get) a variable or value
 - *Setters (ex: setNumHead)*
 - Modify (or set) a variable or value
 - Can be used to set conditions to prevent value from being invalid (ex: prevent numHeads from being set a value less than 0 since we cannot have a negative amount of heads)

What are ALL of the methods we need then?

Public Methods

- Coin Constructor
- getNumHeads
- getNumTails
- flip
- reset

Private Methods

- setNumHeads
 - *Parameters: int*
 - new value for numHeads to be set to
 - *Return Type: void*
- setNumTails
 - *Parameters: int*
 - new value for numTails to be set to
 - *Return Type: void*

What about our members of Coin?

All of our members of Coin will be **private**. This is to protect the members from being accessed and changed outside of Coin. If another file wants to change or view our members, they're going to have to go through our setters or getters to do so.

FINALS

- `int final HEADS = 1`
- `int final TAILS = 0`

Used in flip method to determine if random number is a heads or a tails. Using these finals to compare is more descriptive and easier to read than using just a hardcoded 1 or 0

NON-FINALS

- `Random r`
- `int numHeads`
- `int numTails`

Used to keep track of the number of times we flipped a heads or a tails

Writing Coin.java

- DECLARE all your variables globally so their scope extends to the whole file
- INITIALIZE final variables only

```
import java.util.Random;
public class Coin
{
    // define members up here BUT dont initialize them up here unless they are final
    private final int TAILS=0;
    private final int HEADS=1;
    private Random r;
    private int numHeads,numTails;
```

Writing Coin.java

- Create your Coin constructor
 - *Initialize random object*
 - *Initialize numHeads and numTails*

```
public Coin( int seed )
{
    r = new Random( seed ); // DONE ONLY ONCE FOR EACH NEW COIN
    reset();
}
```

Writing Coin.java

- Create your flip() method
 - *Use random object to determine if heads or tails*
 - *Compare random value to final variables HEADS or TAILS*
 - *Increase numTails or numHeads using set methods*
 - *Return a String or a char value*

```
// FLIP method produced a head or tail with 50/50 odds

public String flip() // COULD OPTIONALLY RETURN A CHAR
{
    int side = r.nextInt(2); // produces 0 or 1
    if (side==HEADS)
    {
        setNumHeads( getNumHeads()+1 );
        return "H";
    }
    else
    {
        setNumTails( getNumTails()+1 );
        return "T";
    }
}
```

Writing Coin.java

- Create get methods
 - *Return the value of numHeads and numTails*

```
public int getNumHeads()  
{  
    return numHeads;  
}  
  
public int getNumTails()  
{  
    return numTails;  
}
```

Writing Coin.java

- Create set methods
 - *Set the value of numHeads and numTails equal to the value being passed in*

```
// NOT CALLED BY MAIN, CALLED ONLY INTERNALLY
private void setNumHeads( int h )
{
    numHeads=h; // COULD OPTIONALLY TEST FOR >=0
}

// NOT CALLED BY MAIN, CALLED ONLY INTERNALLY
private void setNumTails( int t )
{
    numTails=t; // COULD OPTIONALLY TEST FOR >=0
}
```

Writing Coin.java

- Create reset methods
 - *Set the value of numHeads and numTails equal to 0*

```
public void reset()  
{  
    setNumHeads(0);  
    setNumTails(0);  
}
```