

## Program Warm-up for the Day

- Write a program that reads in a digit (in range 0 to 9). The program should reject non-digits, and repeatedly ask for a number until a digit is entered. The program should then print the square of that digit.
- Extra credit: Read in three digits, and print their product.

---

---

---

---

---

---

---

---

```
import java.util.Scanner;  
class Digit {  
    public static void main(String[] args) {  
        Scanner input = new Scanner(System.in);  
  
    }  
}
```

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Remember first class ...

- Recipe for Macaroni and Cheese ...
  - Boil some water
  - Open the box
  - Remove the cheese packet
  - Put the macaroni in the boiling water
  - Boil for 7 minutes
  - Drain water
  - Add butter and cheese powder (from packet)
  - Stir

---

---

---

---

---

---

---

---

## Recipe for three course dinner

- Make a salad
- Make Macaroni and Cheese
- Make apple pie
  
- Recipe for Macaroni and Cheese
  - Boil some water
  - Open the box
  - Remove the cheese packet
  - Put the macaroni in the boiling water
  - Boil for 7 minutes
  - Drain water
  - Add butter and cheese powder (from packet)
  - Stir

---

---

---

---

---

---

---

---

```
class Message {  
  
    public static void main(String[] args) {  
        System.out.println("HELLO Katie!");  
        printMessage();  
        System.out.println("Goodbye.");  
    }  
  
    public static void printMessage() {  
        System.out.println("A message for you: ");  
        System.out.println("Have a nice day!\n");  
    }  
}
```

---

---

---

---

---

---

---

---

## Method call-return

The system calls main().

main() calls System.out.println().  
"HELLO Katie!" is printed  
System.out.println() returns.

Main() calls printMessage().  
printMessage() calls System.out.println().  
"A message for you: " is printed.  
System.out.println() returns.  
printMessage() calls System.out.println().  
"Have a nice day!\n" is printed.  
System.out.println() returns.  
printMessage() returns.

Main() calls System.out.println().  
"Goodbye." is printed.  
System.out.println() returns.

Main() returns to the system.

The program ends.

---

---

---

---

---

---

---

---

---

---

```
class Message2 {  
  
    public static void main(String[] args) {  
        System.out.println("HELLO Katie!");  
        printMessage(5);  
        System.out.println("Goodbye.");  
    }  
  
    public static void printMessage(int n) {  
        System.out.println("A message for you: ");  
        for (int i = 0; i < n; i++)  
            System.out.println("Good day!\n");  
    }  
}
```

---

---

---

---

---

---

---

---

---

---

## Don't forget **static**

Until chapter 6, all of the methods you write should have the qualifier **static**. If you leave it off you will get a message like:

Can't make static reference to method  
*returnType methodName(...)* in class *YourClass*.

---

---

---

---

---

---

---

---

---

---

```
class Min2 {

    public static void main(String[] args) {
        int j = 78, k = 3 * 30;
        System.out.println("Minimum of two integers Test:");
        int m = min(j, k);
        System.out.println("The minimum of : "
            + j + " , " + k + " is " + m);
    }

    public static int min(int a, int b) {
        if (a < b)
            return a;
        else
            return b;
    }
}
```

---

---

---

---

---

---

---

---

## More about return

Control returns *immediately* from a method when a return is executed.

```
public static int min(int a, int b) {
    if (a < b)
        return a;
    else if (b < a)
        return b;

    System.out.println("they are equal!!!");
    return a;
}
```

---

---

---

---

---

---

---

---

## Defining Methods

```
public static ReturnType Identifier ( ParameterList ) {
    Body
}
```

- *ReturnType* is the type of value returned from the method/function.
- *Identifier* is the name of the method/function.
- *ParameterList* is a list of variables (called *formal parameters*) that will be used to pass information into the method.
  
- *Body* is a list of statements and declarations describing the action performed by this method.

---

---

---

---

---

---

---

---

```
class Min2Bad {

    public static void main(String[] args) {
        int j = 78, k = 3 * 30;
        System.out.println("Minimum of two integers Test:");
        int m = min();
        System.out.println("The minimum of : "
            + j + " , " + k + " is " + m);
    }

    static int min() {
        if (j < k)
            return j;
        else
            return k;
    }
}
```

---

---

---

---

---

---

---

---

```
public class SquareRoots2 { // contains scope errors

    public static void main(String[] args) {
        double root = Math.sqrt(99);

        System.out.println("the root of 99 is " + root);

        for (int i = 1; i <= 10; i++) {
            double root = Math.sqrt(i);
            double square = root * root;
            System.out.println("the root of " + i + " is " + root);
            System.out.println("squaring that yields " + square);
        }

        System.out.println("Final value of square is " + square);
    }
}
```

---

---

---

---

---

---

---

---

## Watch for repeating lines.

If you find you have typed the same sequence of 3 or more lines, more than once, then you should consider creating a method to accomplish the task of those repeated lines.

Occasionally, even a one or two line method can add to program readability.

---

---

---

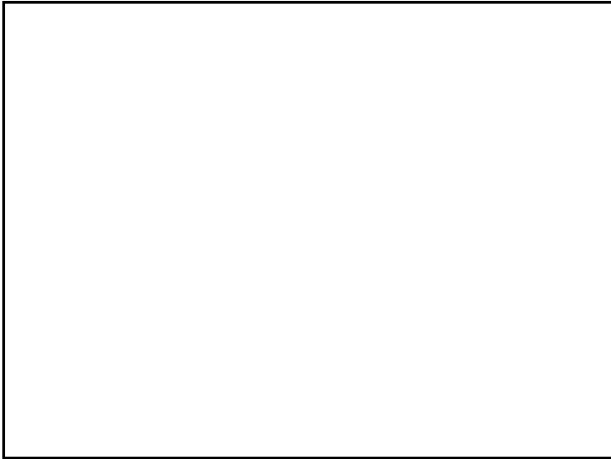
---

---

---

---

---



---

---

---

---

---

---

---

---

## Guide for Tablet PCs

- One tablet per three students
  - need to sign sheet and leave ID
  - must be in first four rows (networking issue)
  - requirement: use tablets at least once
- In-class puzzles
  - *everybody* tries them
  - students with tablets *must* submit
    - functions as class representatives, gives me feedback on what's easy and what's hard
- Tablets gift from HP – we'd like more!

---

---

---

---

---

---

---

---

## Instructions for Tablet PCs

- Plug in (in first four rows)
- Turn on (switch on side)
- Programs > Classroom Presenter
- Role > Student
- Connect > Classroom 1
- ...
- Submit

---

---

---

---

---

---

---

---

## Program Warm-up for the Day

- Write a function called `isPrime` that takes an integer argument, and returns `true` if that integer is prime (and `false` otherwise).
- Write a program that prints all prime numbers from 1 to 1000.

---

---

---

---

---

---

---

---

```
class Prime {  
    public static boolean isPrime(int n) {  
  
  
  
  
  
  
  
  
  
    }  
}
```

---

---

---

---

---

---

---

---

```
class Prime {  
    public static boolean isPrime(int n) { ... }  
  
    public static void main(String[] args) {  
  
  
  
  
  
  
  
  
  
    }  
}
```

---

---

---

---

---

---

---

---

## Method Definitions - Good and Bad

- `public static void printPrompt() { ... }`
- `public static min(int a, int b) { ... }`
- `public static int max(int a, b) { ... }`
- `public static readInt(Scanner input) { ... }`
- `public static int readInt(Scanner input) { ... }`
- `public static int promptAndReadInt(Scanner input, String prompt) { ... }`

---

---

---

---

---

---

---

---

## Method Calls - Good and Bad

- `public static int max(int a, int b) { ... }`
- `min(3, 4);`
- `int m = min(3,false);`
- `if ( min(3,4) == true ) { ... }`
- `int m = min(3, 5);`
- `int x=3; int m = 5 + min(10, x);`

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Programming Challenge (1)

```
/** This method returns the
 * minimum of its two arguments */
public static int min(int a, int b) {

}
}
```

---

---

---

---

---

---

---

---

## Programming Challenge (2)

```
/** This method prints the prompt string,
 * and returns an int read from the scanner. */
public static int promptAndReadInt(
    Scanner input, String prompt) {

}
}
```

---

---

---

---

---

---

---

---

## Programming Challenge (3)

```
import java.util.Scanner;
public class Min {
    public static void main(String[] args) {
        Scanner input = new Scanner();

    }
}
}
```

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Design

- Design
  - How to write a program (or pseudocode) to implement a given specification
  
- Top-down design
  - Repeatedly decompose the problem into smaller problems

---

---

---

---

---

---

---

---

## Simulating a Coin Toss

Problem: What is the probability that you can toss some number,  $n$ , heads in a row?

Pseudocode:

```
Input the number of heads in a row for a trial.  
Input the number of trials.  
Perform the specified number of trials.  
Print the result.
```

---

---

---

---

---

---

---

---

### Pseudocode for "performing the n trials"

```
initialize the number of successes to 0
while there are more trials to run
  run one trial
  if the trial was a success
    increment the number of successes
end while loop
return the number of successful trials
```

---

---

---

---

---

---

---

---

### Pseudocode for "performing one trial"

```
let numTosses be the number of tosses per trial
initialize the number of tosses done to zero
while number of tosses done is less than numTosses
  toss the coin
  if the coin comes up tails
    return failure
  increment the number of tosses done
end while loop
return success
```

---

---

---

---

---

---

---

---

```
// Perform one trial.
// return true if numTosses heads are tossed in a row

public static boolean isAllHeads(int numTosses) {

}

}
```

---

---

---

---

---

---

---

---

```
// Perform one trial.
// return true if numTosses heads are tossed in a row

public static boolean isAllHeads(int numTosses) {

    for(int numDone = 0; numDone < numTosses; numDone++) {

        double outcome = Math.random(); // toss the coin
        if ( outcome < 0.5)
            return false; // tossed a tail
        }

    return true; // tossed all heads

}
```

---

---

---

---

---

---

---

---

```
// perform numTrials simulated coin tosses
// and return the number of successes

public static int performTrials(int numHeads,
                                int numTrials) {

}

}
```

---

---

---

---

---

---

---

---

```
// perform numTrials simulated coin tosses
// and return the number of successes

public static int performTrials(int numHeads,
                                int numTrials) {

    System.out.println("Performing " + numTrials
        + " trials for " + numHeads + " heads in a row");
    int numSuccesses = 0;
    for (int trials= 0 ; trials < numTrials; trials++)
        // perform one trial
        if (isAllHeads(numTosses)) {
            numSuccesses++; // trial was a success
        }
    return numSuccesses;
}

}
```

---

---

---

---

---

---

---

---

```
class CoinToss {
    public static void main(String[] args) {
        //Input the number of heads in a row to try for.
        int numHeads = 4;        //Just use 4 for testing

        //Input the number of trials to run.
        int numTrials = 10000;  //Use 10000 for testing

    }
}
```

---

---

---

---

---

---

---

---

```
class CoinToss {
    public static void main(String[] args) {
        //Input the number of heads in a row to try for.
        int numHeads = 4;        //Just use 4 for testing

        //Input the number of trials to run.
        int numTrials = 10000;  //Use 10000 for testing

        //Perform the specified number of trials
        int numSuccesses = performTrials(numHeads,numTrials);

        //Print the results
        System.out.println(
            "Got " + numHeads + " heads in a row on "
            + numSuccesses + " out of " + numTrials + " trials")
    }
}
```

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Program Warm-up for the Day

- Write a program that reads in lines from the console and prints them out, stopping when the input "quit" is read in.

```
import java.util.Scanner;
class Cat {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

    }
}
```

---

---

---

---

---

---

---

---

## Program Warm-up for the Day (2)

- Write a program that reads in lines from the console and prints them out, stopping when the *end-of-file* is reached.

```
import java.util.Scanner;
class Cat {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

    }
}
```

---

---

---

---

---

---

---

---

## null in Strings

- Write a program that reads in lines from the console and prints them out, stopping when the *end-of-file* is reached.

```
class Null {
    public static void main(String[] args) {
        String a;
        a = "Hello";
        System.out.println("a: "+a);
        a = null;
        System.out.println("a: "+a);
        if (a == null ) {
            System.out.println("a is null");
        }
    }
}
```

---

---

---

---

---

---

---

---

## Next Class: Blackjack

- Sample run

```
> java Blackjack
Please enter points in the dealers up card
6
Please enter your points so far
17
Hitting wins on 94 out of 1000 trials
Staying wins on 34 out of 1000 trials
>
```

---

---

---

---

---

---

---

---

## Blackjack Rules

- Points
  - Aces worth 1 point
  - number cards worth their number
  - face cards worth 10 points
- Play
  - dealer gets one face-up card
  - you can
    - keep hitting (getting more cards/points)
    - stay

---

---

---

---

---

---

---

---

## Blackjack Rules

- Goal: get close to 21, but not over
- Play
  - dealer gets one face-up card
  - you can
    - hit (getting more cards/points)
      - you lose if over 21 points
    - stay
  - dealer gets more cards till at least 17 points
    - dealer busts if over 21 points
  - You win if you have more points than the dealer

---

---

---

---

---

---

---

---

## Simulating Blackjack

- Should we “hit” or should we “stay”, given the cards on the table?
  - Input what points the dealer has
  - Input what points we have
  - Compute how many times out of 10000 trials we win if we “hit”
  - Compute how many times out of 10000 trials we win if we “stay”
  - Print results

---

---

---

---

---

---

---

---

## Pseudocode for “Run 10000 Trials”

- Input: dealers points, our points, and whether we hit or stay
- initialise number of successes to 0
- for each trial, if trial is successful, increment the number of successes

---

---

---

---

---

---

---

---

## Pseudocode for “Run 1 Trials”

- Input: dealers points, our points, and whether we hit or stay
- if we hit, add a random card to our hand
  - over 21 points => we lose
- dealer hits till at least 17 points
  - over 21 points => we win
- we win if more points than dealer

---

---

---

---

---

---

---

---

## Pseudocode for "Get a random card"

- Get a random number between 0 and .9999999 using Math.random()
- Multiply by 13
- Add 1
- Convert to an int between 1 and 13
- if number is > 10 then return 10 points, else return that number

---

---

---

---

---

---

---

---

## Code for "Run 1 Trial"

```
public static boolean performOneTrial(  
    int dealersPoints, int myPoints, boolean hit) {  
  
  
  
  
  
  
  
  
  
}
```

---

---

---

---

---

---

---

---

## Code for "Run 10000 Trial"

```
public static boolean performTrials(  
    int dealersPoints, int myPoints, boolean hit) {  
  
  
  
  
  
  
  
  
  
}
```

---

---

---

---

---

---

---

---

## Code for "Blackjack"

```
import java.util.Scanner;
class Blackjack {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

    }
}
```

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Program Warm-up for the Day

- Write a method that takes two integer arguments and returns the larger of those two arguments.

---

---

---

---

---

---

---

---

```
class ByValue {
    public static void main(String[] args) {
        int i = 10;
        int j = myPrint(i);
        System.out.println("i="+i+", j="+j);
    }

    public static int myPrint(int k) {
        System.out.println("k="+k);
        k = k * 2;
        System.out.println("k="+k);
        return k;
    }
}
```

---

---

---

---

---

---

---

---

```
class ByValue {
    public static void main(String[] args) {
        int i = 10;
        int j = myPrint(i);
        System.out.println("i="+i+", j="+j);
    }

    public static int myPrint(int i) {
        System.out.println("i="+i);
        i = i * 2;
        System.out.println("i="+i);
        return i;
    }
}
```

---

---

---

---

---

---

---

---

## Method Overloading

- Simple idea: The method called is determined by:
  - the name of the method, and
  - the number and type of parameters in the call
- So, two methods can have the same name as long as they have different numbers and/or types of parameters

---

---

---

---

---

---

---

---

```
class Test {
    public static void main(String[] args) {
        int i=1,j=2;
        System.out.println( min(i,j) );
        double c=4.3, d=5.2;
        System.out.println( min(c,d) );
    }

    public static int min(int i, int j) {
        if (i < j) return i;
        else return j;
    }

    public static double min(double a, double b) {
        if (a < b) return a;
        else return b;
    }
}
```

---

---

---

---

---

---

---

---

## An ambiguous overload.

```
//AmbiguousOverload.java: won't compile
class AmbiguousOverload {
    public static void main(String[] args) {
        int i = 1, j = 2;
        System.out.println( lessThan(i, j) );
    }

    static boolean lessThan(double x, int y) {
        return x < y;
    }

    static boolean lessThan(int x, double y) {
        return x < y;
    }
}
```

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Recursion

```
public class Recur {
    public static void main(String[] args) {
        sayGoodBye(5);
    }

    static void sayGoodBye(int n) {
        if (n < 1) {
            System.out.println("#####");
        } else {
            System.out.println("Goodbye n=" + n);
            sayGoodBye(n - 1);
        }
    }
}
```

---

---

---

---

---

---

---

---

## Recursion

- When a method calls itself, this is referred to as *recursion*
- Recursion can be confusing, but is extremely powerful
- Often used when a mathematical operation is defined in terms of other values of itself
  - Examples: factorials, fibonacci numbers, ...

---

---

---

---

---

---

---

---

## Form of a Recursive Function

```
public static <type> recursiveMethod(<args>) {
    if (<stopping condition>) {
        // whatever you do at the end
    } else {
        recursiveMethod(<different args>);
    }
}
```

---

---

---

---

---

---

---

---

## Example: Factorial

- $n! = n * (n-1) * (n-2) * \dots * 2 * 1$
- $n! = n * (n-1)!$
- Recall:  $0! = 1$  and  $1! = 1$

```
public static int factorial(int n) {
    if(n <= 1) {
        return 1;
    } else {
        return n * factorial(n-1);
    }
}
```

---

---

---

---

---

---

---

---

## Example: Factorial (cont.)

- Suppose we execute factorial(4)
  - main calls factorial(4) <a>
  - <a> calls factorial(3) <b>
  - <b> calls factorial(2) <c>
  - <c> calls factorial(1) <d>
  - <d> returns 1
  - <c> returns  $2 * 1 (= 2)$
  - <b> returns  $3 * 2 (= 6)$
  - <a> returns  $4 * 6 (= 24)$
  - and that is the answer: 24

---

---

---

---

---

---

---

---

## Your Programming Challenge

- Define a method “exp” that computes  $x^n$ 
  - Remember that  $x^0 = 1$
  - and that  $x^{n+1} = x * x^n$

```
public static double exp(double x, int n) {
```

```
}
```

---

---

---

---

---

---

---

---

## Example: Fibonacci numbers

- Each Fibonacci number is defined as the sum of the two previous Fibonacci numbers
- $\text{fibonacci}(n) = \text{fibonacci}(n-1) + \text{fibonacci}(n-2)$
- $\text{fibonacci}(0) = 1, \text{fibonacci}(1) = 1$

```
public static int fibonacci(int n) {
```

```
}
```

---

---

---

---

---

---

---

---

## Example: Fibonacci (cont.)

- Suppose we execute `fibonacci(3)`
  - main calls `fibonacci(3)` <a>
  - <a> calls `fibonacci(2)` <b> and `fibonacci(1)` <c>
  - <b> calls `fibonacci(1)` <f> and `fibonacci(0)` <g>
  - <f> returns 1
  - <g> returns 1
  - <c> returns 1
  - <b> returns 2
  - <a> returns 3
  - and that is the answer: 3

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

- Twenty-One Pickup is a two-player game that starts with a pile of 21 stones. Each player takes turns removing 1, 2, or 3 stones from the pile. The player that removes the last stone wins.

- Is there an *optimal strategy* for playing Twenty-One Pickup?
- Can we write a program to figure out this optimal strategy?
- Challenge: Can you beat the professor?

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

```
class Pickup {  
    // Alana's turn - will she win?  
    public static boolean willAlanaWin(int stones) {  
  
    }  
    // Cormac's turn - will he win?  
    public static boolean willCormacWin(int stones) {  
  
    }  
}
```

---

---

---

---

---

---

---

---

## Testing

- At a minimum you want to
  - Execute every instruction at least once
  - Take every branch at least once
- Also, try every possible input
  - valid and invalid
  - not always possible
  - do the best you can

---

---

---

---

---

---

---

---