CS 106A, Lecture 8
Return; Boolean Logic

reading:

Art & Science of Java, 5.2 - 5.3; 6.1

Based on slides created by Keith Schwarz, Mehran Sahami, Eric Roberts, Stuart Reges, and others.

Lecture at a glance

- Today we will learn about **return values**.
  - Parameters send values *into* a method;
    returns send a value *out* of a method.
  - Powerful tool for decomposition and reusable methods.
- We will also learn about the **`boolean`** data type.
  - Stores logical true/false values.
  - Allows us to make methods that can be used as logical tests.

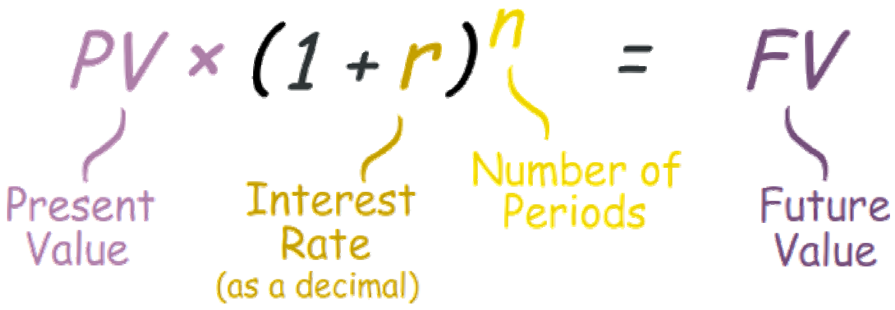
Investment exercise #2

- Suppose our **Investment** program needs to display the difference in profit from the two investments. *(The new last line of output.)*

```
Investor #1:
Initial amount? 100.00
Interest rate%? .03
Num. of months? 5
Final amount = $115.93
Profit = $15.93 (16%)
medium

Investor #2:
Initial amount? 5.25
Interest rate? .08
Num. of months? 24
Final amount = $33.29
Profit = $28.04 (534%)
strong

Profit difference = $12.11
```



Profit	Category
0 - 10%	weak
10 - 50%	medium
over 50%	strong

Java's Math class

Method name	Description
Math.abs(***value***)	absolute value
Math.ceil(***value***)	rounds up
Math.floor(***value***)	rounds down
Math.log(***value***)	logarithm, base *e*
Math.log10(***value***)	logarithm, base 10
Math.max(***value1, value2***)	larger of two values
Math.min(***value1, value2***)	smaller of two values
Math.pow(***base, exp***)	*base* to the *exp* power
Math.round(***value***)	nearest whole number
Math.sqrt(***value***)	square root

Math.sin(***value***)
Math.cos(***value***)
Math.tan(***value***)
	sine/cosine/tangent of an angle in radians

Math.toDegrees(***value***)
Math.toRadians(***value***)
	convert degrees to radians and back

Constant	Description
Math.E	2.7182818...
Math.PI	3.1415926...

No output?

- Simply calling these methods produces no visible result.

```
Math.pow(3, 4);     // no output
```

- Math methods use a Java feature called *return values* that cause them to be treated as expressions.

- The program runs the method, computes the answer, and then "replaces" the call with its computed result value.

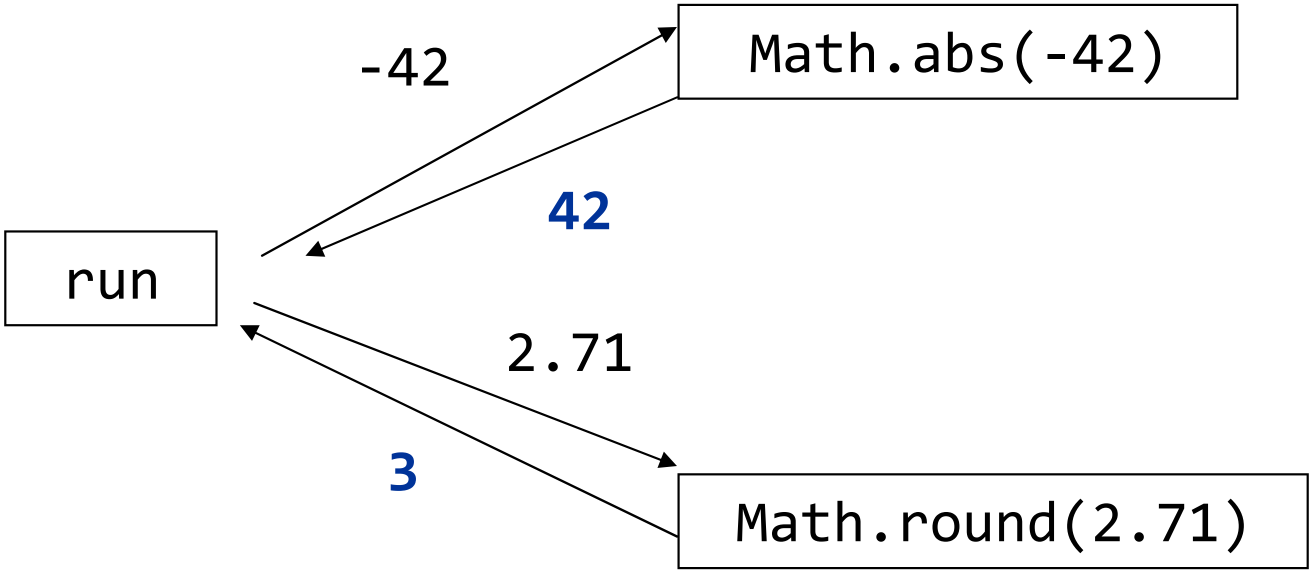
```
~~Math.pow(3, 4)~~;     // no output
81.0;               // no output
```

- To see the result, we must print it or store it in a variable.

```
double result = Math.pow(3, 4);
println(result);    // 81.0
```

Return

- **return**: To send out a value as the result of a method.
  - Parameters send information *in* from the caller to the method.
  - Return values send information *out* from a method to its caller.
    - A call to the method can be used as part of an expression.



- **Q:** Why return? Why not just println the result value?

Methods that return

```
public type name(parameters) {
    statements;
    ...
    return expression;
}
```

- Example:

```
// Returns the slope of the line between the given points.
public double slope(int x1, int y1, int x2, int y2) {
    double dy = y2 - y1;
    double dx = x2 - x1;
    return dy / dx;
}
```

`slope(7, 11, 5, 2)` returns `4.5`

Common error: Not storing

- Many students incorrectly think that a `return` statement sends a variable's name back to the calling method. (The value is sent.)

```
public void run() {
    slope(0, 0, 6, 3);
    println("The slope is " + result);  // ERROR:
}                   // cannot find symbol: result

public double slope(int x1, int x2, int y1, int y2) {
    double dy = y2 - y1;
    double dx = x2 - x1;
    double result = dy / dx;
    return result;
}
```

Fixing the common error

- Returning sends the variable's *value* (not name) back.
  Store the returned value into a variable or use it in an expression.

```
public void run() {
    double s = slope(0, 0, 6, 3);
    println("The slope is " + s);
}

public double slope(int x1, int x2, int y1, int y2) {
    double dy = y2 - y1;
    double dx = x2 - x1;
    double result = dy / dx;
    return result;
}
```

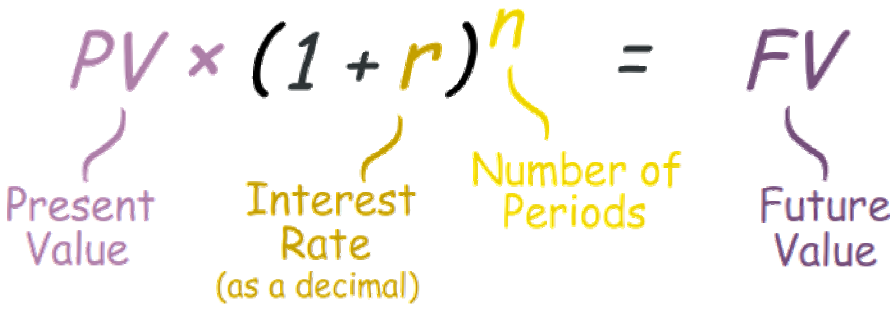
Investment exercise #2

- Modify our **Investment** program to use Math and returns.
  (Note the new last line of output.)

```
Investor #1:
Initial amount? 100.00
Interest rate%? .03
Num. of months? 5
Final amount = $115.93
Profit = $15.93 (16%)
medium

Investor #2:
Initial amount? 5.25
Interest rate? .08
Num. of months? 24
Final amount = $33.29
Profit = $28.04 (534%)
strong

Profit difference = $12.11
```



Profit	Category
0 - 10%	weak
10 - 50%	medium
over 50%	strong

Boolean Logic

Based on slides created by Keith Schwarz, Mehran Sahami, Eric Roberts, Stuart Reges, and others.

Type boolean

- **boolean**: A logical type whose values are `true` and `false`.
  - a `boolean` variable stores the result of a logical test
  - `boolean` can be passed as a parameter or returned

```
boolean minor    = age < 21;
boolean isProf   = iq > 180;
boolean lovesCS = true;

// allow only CS-loving students over 21
if (minor || isProf || !lovesCS) {
    println("Can't enter the club!");
}
```

Boolean return

- You can write a method that returns a `boolean` value:

```
// Returns true if n is even, false if odd.
public boolean isEven(int n) {
    if (n % 2 == 0) {
        return true;
    } else {
        return false;
    }
}
```

- Calls to methods returning `boolean` can be used as tests:

```
if (isEven(42)) { ...
```

  – Karel methods like `frontIsClear`, `beepersPresent` return `boolean`.

Boolean Zen

- Methods that return boolean often have an if/else:

```
// Returns true if both of the numbers passed are odd.
public boolean allOdd(int a, int b, int c) {

    if (a % 2 != 0 && b % 2 != 0 && c % 2 != 0) {
        return true;
    } else {
        return false;
    }
}
```

- But the code above is unnecessarily verbose.

A boolean variable

- We could store the result of the logical test in a variable.

```
// Returns true if both of the numbers passed are odd.
public boolean allOdd(int a, int b, int c) {
    boolean test = a % 2 != 0 && b % 2 != 0 && c % 2 != 0;
    if (test == true) {
        return true;
    } else {
        return false;
    }
}
```

- Notice: Whatever `test` is, we want to return that.
  - If test is `true` , we want to return `true`.
  - If test is `false`, we want to return `false`.

Zen solution

- Observation: The `if/else` is unnecessary.

```
// Returns true if both of the numbers passed are odd.
public boolean allOdd(int a, int b, int c) {
    boolean test = a % 2 != 0 && b % 2 != 0 && c % 2 != 0;
    return test;
}
```

- An even shorter version:
  - We don't need the variable; just test and return in one step.

```
public boolean allOdd(int a, int b, int c) {
    return a % 2 != 0 && b % 2 != 0 && c % 2 != 0;
}
```

Boolean Zen

- Students often test if a result is equal to `true`:

```
if (isEven(54) == true) {     // bad
    ...
}
```

- But this is unnecessary and redundant. Preferred:

```
if (isEven(54)) {             // good
    ...
}
```

- A similar pattern can be used for a false test:

```
if (isEven(57) == false) {    // bad
if (!isEven(57)) {            // good
```

De Morgan's Law

- **De Morgan's Law:** Rules used to negate `boolean` tests.
  - Useful when you want the opposite of an existing test.

Original Expression	Negated Expression	Alternative
`a && b`	`!a || !b`	`!(a && b)`
`a || b`	`!a && !b`	`!(a || b)`

- Example:

Original Code	Negated Code

`if (x == 7 && y > 3) {`
`    ...`
`}`
	`if (x != 7 || y <= 3) {`
	`    ...`
	`}`

isPrime exercise

- Write a method **`isPrime`** that returns `true` if a given integer is a prime number, meaning that it is divisible by only 1 and itself.
  - The first prime number is defined to be 2.
  - For example:
    `isPrime(17)` should return `true`
    `isPrime(24)` should return `false`

- It should be possible to use your method as a logical test:

```
if (isPrime(57)) { ...
```

- *Bonus:* Write a method **`isPerfectNumber`** that returns `true` if a given integer is a "perfect number", which means that it is the sum of its factors, such as 28 = 1 + 2 + 4 + 7 + 14.

Exercise solutions

```java
// Returns whether n's only factors are 1 and itself.
public boolean isPrime(int n) {
    int factors = 0;
    for (int i = 1; i <= n; i++) {
        if (n % i == 0) {
            factors++;
        }
    }
    return factors == 2;
}

// Returns whether n equals the sum of its factors.
public boolean isPerfectNumber(int n) {
    int sum = 0;
    for (int i = 1; i < n; i++) {
        if (n % i == 0) {
            sum += i;
        }
    }
    return sum == n;
}
```

Randomization

RandomGenerator

- import acm.util.*;

Method	Description
RandomGenerator. getInstance() or, new RandomGenerator()	creates a new RandomGenerator object
rg.nextBoolean() ***rg***.nextBoolean(***prob***)	randomly returns true or false, using a random real-number probability from 0-1
rg.nextColor()	a randomly created RGB color *(used later)*
rg.nextDouble() ***rg***.nextDouble(***low***, ***hi***)	a random real number in the given range, or in range 0.0 - 1.0 if no range provided
rg.nextInt(***low***, ***hi***)	a random integer in the given range, inclusive

```
RandomGenerator rg = RandomGenerator.getInstance();
int randomDigit = rg.nextInt(0, 9);
println(randomDigit);
```

Other uses of random

- `nextInt` works on a continuous range, but you can simulate non-continuous ranges with * and other operators.
  - Example: Get a random *odd* integer between 1 and 99 inclusive:

    ```
    int odd = 1 + rg.nextInt(0, 49) * 2;
    ```

- **Q:** How could you choose a random non-integer value?
  - Example: How would you choose to randomly play Rock-Paper-Scissors by randomly selecting Rock, Paper, or Scissors each turn?

Rock-paper-scissors

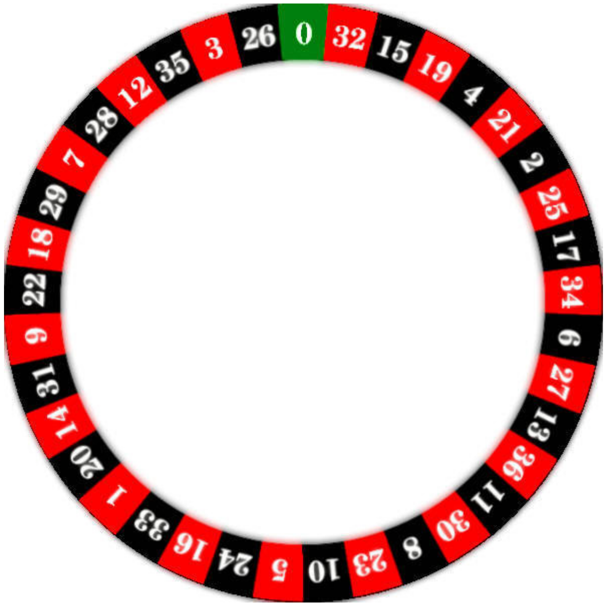
- Any set of possible values can be mapped to integers.
  - Example: Code to randomly play Rock-Paper-Scissors:

```
int r = RandomGenerator.getInstance().nextInt(1, 3);
if (r == 1) {
    println("Rock");
} else if (r == 2) {
    println("Paper");
} else {  // r == 3
    println("Scissors");
}
```

Random roulette

- Write a console program **`Roulette`** that simulates the gambling game of Roulette, with the following characteristics:
  - The player begins with $10 and bets (up to) $3 per spin of the wheel.
  - If the wheel comes up 1-18, the player wins $3. Else, player loses $3.
  - Play until the player gets $1000 or drops to $0. Print the max money.

```
bet $3, spin 15, money = $13
bet $3, spin 35, money = $10
bet $3, spin 7, money = $13
bet $3, spin 4, money = $16
bet $3, spin 28, money = $13
bet $3, spin 19, money = $10
bet $3, spin 21, money = $7
bet $3, spin 26, money = $4
bet $3, spin 36, money = $1
bet $1, spin 22, money = $0
max = $16
```



When to return?

- Methods with loops and return values can be tricky.
  - When and where should the method return its result?

- Write a method named **`seven`** that draws up to ten random lotto numbers from 1-30.
  - If any of the numbers is a lucky 7, the method should stop and return `true`. If none of the ten are 7 it should return `false`.
  - The method should print each number as it is drawn.

```
15 29 18 29 11 3 30 17 19 22        (first call)
29 5 29 4 7                          (second call)
```

Seven solution?

- **Q:** What is the behavior of this solution?

```
public boolean seven() {
    for (int i = 1; i <= 10; i++) {
        int num = RandomGenerator.getInstance()
                                 .nextInt(1, 30);
        print(num + " ");
        if (num == 7) { return true;  }
        else          { return false; }
    }
}
```

- **A.** It works fine.
- **B.** It always returns immediately after the first draw.
- **C.** It behaves incorrectly if the first roll is a 7.
- **D.** It behaves incorrectly if the last roll is a 7.
- **E.** Other

Return at proper time

```
// Draws 10 lotto numbers; returns true if one is 7.
public boolean seven() {
    RandomGenerator randy = RandomGenerator.getInstance();
    for (int i = 1; i <= 10; i++) {
        int num = randy.nextInt(1, 30);
        print(num + " ");
        if (num == 7) {    // found lucky 7; can exit now
            return true;
        }
    }
    return false;   // if we get here, there was no 7
}
```

- Returns `true` immediately if 7 is found.
  - If 7 isn't found, the loop continues drawing lotto numbers.
  - If all ten aren't 7, the loop ends and we return `false`.

Overflow (extra) slides

Return examples

sumOfRange
factorial
ReturnMystery1

- Methods with returns are often like math functions or formulas:

```
// Converts degrees Fahrenheit to Celsius.
public double fToC(double degreesF) {
    double degreesC = 5.0 / 9.0 * (degreesF - 32);
    return degreesC;
}

// Computes triangle hypotenuse length given its side lengths.
public double hypotenuse(int a, int b) {
    double c = Math.sqrt(a * a + b * b);
    return c;
}
```

- You can shorten the examples by returning an expression:

```
public double fToC(double degreesF) {
    return 5.0 / 9.0 * (degreesF - 32);
}
```

Procedural design

- General rules for designing good methods:

1. Each method should have a clear set of responsibilities.

2. No one method should do too large a share of the overall task.

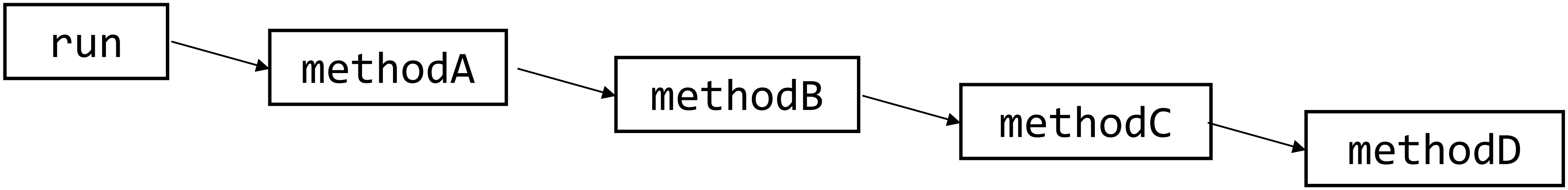
3. Minimize coupling and dependencies between methods.

4. The `run` method should be a concise summary of the overall program.

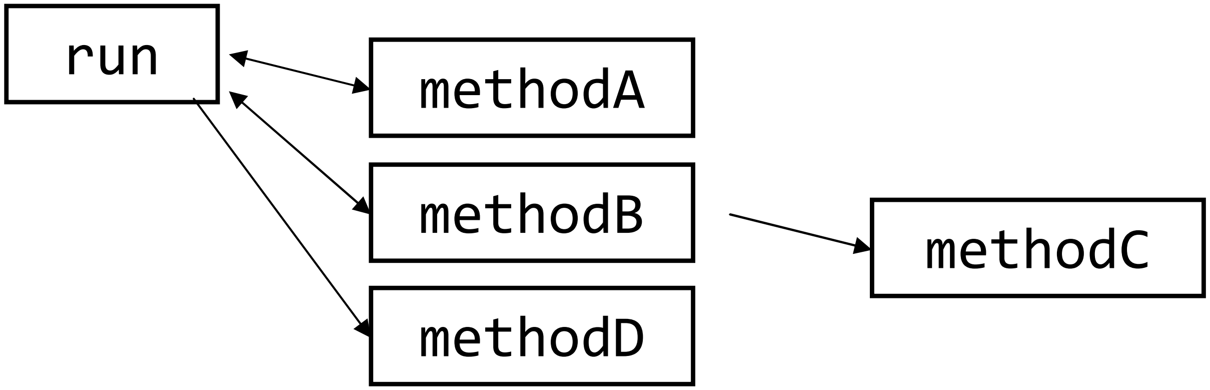
5. Data should be declared/used at the lowest level possible.

"Chaining"

- **run** should be a concise summary of your program.
  - It is bad if each method calls the next without ever returning (we call this *chaining*):



- A better structure has **run** make most (not all!) of the calls.
  - Methods must return values to run to be passed on later.



BMI 2 exercise

Modify our body mass index (BMI) program to use returns.

$$BMI = \frac{weight}{height^2} \times 703$$

BMI	Category
below 18.5	class 1
18.5 - 24.9	class 2
25.0 - 29.9	class 3
30.0 and up	class 4

- Write the following program:

```
Enter Person 1's information:
height (in inches)? 70.0
weight (in pounds)? 194.25

Enter Person 2's information:
height (in inches)? 62.5
weight (in pounds)? 130.5

Person 1 BMI = 27.868928571428572
class 3
Person 2 BMI = 23.485824
class 2

Difference = 4.383104571428515
```

BMI bad chained solution

```
// This is not a good model to follow!
import acm.program.*;

public class BMI extends ConsoleProgram {
    public void run() {
        getHeightWeight(1);
    }

    public void getHeightWeight(int number) {
        println("Enter Person " + number + "'s information:");
        double height = readDouble("height (in inches)? ");
        double weight = readdouble("weight (in pounds)? ");
        calcBmi(number, height, weight);
    }

    public void calcBmi(int number, double height, double weight) {
        double bmi = weight * 703 / height / height;
        println("BMI = " + bmi);
        weightClass(number, bmi);
    }

    ...
```

BMI bad solution, cont'd.

```
// This is not a good model to follow!

    public void weightClass(int number, double bmi) {
        println("Person " + number + " BMI = " + bmi);
        if (bmi < 18.5) {
            println("class 1");
        } else if (bmi < 25.0) {
            println("class 2");
        } else if (bmi < 30.0) {
            println("class 3");
        } else {
            println("class 4");
        }

        if (number == 1) {
            getHeightWeight(2);   // do the second person
        }
    }
}
```

Boolean exercises

- **`hasAnOddDigit`** : returns `true` if any digit of an integer is odd.
  - `hasAnOddDigit(4822116)` returns true
  - `hasAnOddDigit(2448)` returns false

- **`allDigitsOdd`** : returns `true` if every digit of an integer is odd.
  - `allDigitsOdd(135319)` returns true
  - `allDigitsOdd(9174529)` returns false

- **`isAllVowels`** : returns `true` if every char in a String is a vowel.
  - `isAllVowels("eIeIo")` returns true
  - `isAllVowels("oink")` returns false
  - (try this one after Friday's lecture!)

Random exercise

- **Q:** Which best describes the result of the following call?

```
int n = rg.nextInt(0, 50) * 2 + 1;
```

A. a random integer between 1 and 100 inclusive

B. a random integer between 1 and 150 inclusive

C. a random even integer between 2 and 50 inclusive

D. a random odd integer between 1 and 101 inclusive

E. n/a

Random dice exercise

RollTwoDice

- Write a console program **RollTwoDice** that simulates rolling of two 6-sided dice until their combined result comes up as 7.

```
2 + 4 = 6
3 + 5 = 8
5 + 6 = 11
1 + 1 = 2
4 + 3 = 7
You win!
```

Random dice solution

```
import acm.program.*;
import acm.util.*;

public class DiceRoller extends ConsoleProgram {
    public void run() {
        RandomGenerator rg = RandomGenerator.getInstance();
        int tries = 0;
        int sum = 0;
        while (sum != 7) {
            // roll the dice once
            int roll1 = rg.nextInt(1, 6);
            int roll2 = rg.nextInt(1, 6);
            sum = roll1 + roll2;
            println(roll1 + " + " + roll2 + " = " + sum);
            tries++;
        }
        println("You win!");
    }
}
```