Computer Science: Create Your Own RPG Day #2

OPPTAG Explorations 2014

Brian Nakayama[1]

[1] Department of Computer Science, Iowa State University, Ames, IA 50010, USA

July 8th, 2014

Review #1

```
int i = 30 % 7;
```

Review #1

```
int i = 30 % 7;  \\ 2
int j = 30 / 5;
```

Review #1

```
int i = 30 % 7;  \\ 2
int j = 30 / 5;  \\ 6
int k = i * i - j;
```

Review #1

```
int i = 30 % 7;  \\ 2
int j = 30 / 5;  \\ 6
int k = i * i - j;  \\ -2
int h = 0;
if(k <= 0){
   h += 4;
```

Review #1

```
int i = 30 % 7;  \\ 2
int j = 30 / 5;  \\ 6
int k = i * i - j;  \\ -2
int h = 0;
if(k <= 0){
   h += 4;   \\ 4
   if(k == 0){
      h /= 1;
   } else if (!(k >= -1)){
      h /= 2;
   } else {
      h /= 4;
   }
```

Review #1

```
int i = 30 % 7;  \\ 2
int j = 30 / 5;  \\ 6
int k = i * i - j;  \\ -2
int h = 0;
if(k <= 0){
   h += 4;   \\ 4
   if(k == 0){
      h /= 1;
   } else if (!(k >= -1)){
      h /= 2;
   } else {
      h /= 4;
   }  \\ 2
```

Review #1

```
if(k == 0){
   h /= 1;
} else if (!(k >= -1)){
   h /= 2;
} else {
   h /= 4;
} \\ 2
System.out.print("Zombies eat bra");
if((false || k == -2)&& !(j*h*i > 20)){
   System.out.print("i");
}
System.out.print("ns and oats.")
```

Review #1

```
if(k == 0){
   h /= 1;
} else if (!(k >= -1)){
   h /= 2;
} else {
   h /= 4;
} \\ 2
System.out.print("Zombies eat bra");
if((false || k == -2)&& !(j*h*i > 20)){
   System.out.print("i");
}
System.out.print("ns and oats.") \\ brans
```

Review #2

```
boolean b; int i = 1; int j = 0;
String s = i + "2" + j + "1" + j;
```

Review #2

```
boolean b; int i = 1; int j = 0;
String s = i + "2" + j + "1" + j;  \\ "12020"
double d = Double.parseDouble(s);
double w = (d + i) / d;
if (w <= 0){
    b = true;
    s+="hello!"
} else {
    b = false;
    s+="goodbye!"
}
```

Review #2

```
boolean b; int i = 1; int j = 0;
String s = i + "2" + j + "1" + j;  \\ "12020"
double d = Double.parseDouble(s);
double w = (d + i) / d;
if (w <= 0){
    b = true;
    s+="hello!"
} else {
    b = false;
    s+="goodbye!"
}  \\ "goodbye!"
```

Review #2

```
if (w <= 0){
    b = true;
    s+="hello!"
} else {
    b = false;
    s+="goodbye!"
} \\ "goodbye!"
if(!(!s.equals("12010goodbye") || b ) && !b){
    System.out.println("Goodbye!")
} else if (!(!b)){
    System.out.println("Hello!")
}
```

Review #2

```
if (w <= 0){
    b = true;
    s+="hello!"
} else {
    b = false;
    s+="goodbye!"
} \\ "goodbye!"
if(!(!s.equals("12010goodbye") || b ) && !b){
    System.out.println("Goodbye!")
} else if (!(!b)){
    System.out.println("Hello!")
}  \\ no print.
```

Eclipse

When writing programs, errors are inevitable, and sometimes frustrating. Syntax errors can be especially frustrating, but luckily there is a tool that highlights such errors:

- Create a folder on your flash drive called MyProjects.

Open Eclipse...

From now on we will use Eclipse

- Eclipse will ask you to select a workspace, select the MyProjects folder you just created. You must select this as your workspace everytime you use Eclipse.
- To start a java project click File→New→Project. Then select Java Project in the Java folder and then click Next.
- Give your project a name, then click Finish

From now on we will use Eclipse

- Eclipse will ask you to select a workspace, select the MyProjects folder you just created. You must select this as your workspace everytime you use Eclipse.
- To start a java project click File→New→Project. Then select Java Project in the Java folder and then click Next.
- Give your project a name, then click Finish
- To create a *.java file, click File→New→Class. Click Finish. Write a simple program that prints out a line.

From now on we will use Eclipse

- Eclipse will ask you to select a workspace, select the MyProjects folder you just created. You must select this as your workspace everytime you use Eclipse.
- To start a java project click File→New→Project. Then select Java Project in the Java folder and then click Next.
- Give your project a name, then click Finish
- To create a *.java file, click File→New→Class. Click Finish. Write a simple program that prints out a line.
- There are two ways to run your code. Either click “Run”, *the green circle with an arrow in it*, or Right Click→Run As→Java Application.

Output

- The output should appear in a terminal in eclipse at the bottom of the screen.
- By default, errors and problems with your code should also show up here.

Output

- The output should appear in a terminal in eclipse at the bottom of the screen.
- By default, errors and problems with your code should also show up here.
- Other Benefits:
    - Syntax highlighting.
    - Eclipse has code completion. Try typing "`System.`" . You should see a list of options. Click on any one of these options to learn more about it. Press enter to insert an option.

Output

- The output should appear in a terminal in eclipse at the bottom of the screen.
- By default, errors and problems with your code should also show up here.
- Other Benefits:
    - Syntax highlighting.
    - Eclipse has code completion. Try typing "`System.`" . You should see a list of options. Click on any one of these options to learn more about it. Press enter to insert an option.

Some more Benefits

- Error correction: Type in the following: “`System.out.pintln("Hello");`. You should see a lightbulb on the left. Click it. The lightbulb offers suggestions as to how to fix your code. Click Change to 'println(..)'.

Some more Benefits

- Error correction: Type in the following: "`System.out.pintln("Hello");`. You should see a lightbulb on the left. Click it. The lightbulb offers suggestions as to how to fix your code. Click Change to 'println(..)'.
- Auto tabbing: Remove or add white space to your codes (more tabs, less tabs, etc.). Hold `Ctrl+Shift+F`. Magic!

Back to String[] args

- What is that `[]`?
- The `[]` *declares* that args is an array.
- An array is a fixed list of variables that all have the same type.

Back to String[] args

- What is that `[]`?
- The `[]` *declares* that args is an array.
- An array is a fixed list of variables that all have the same type.
- For example, when running `java ArgProgram hi there matey !`

```
args[0] = "hi";
args[1] = "there";
args[2] = "matey";
args[3] = "!";
```

Arrays

- To declare an array you use:

```
<type>[] <name>;
```

- Arrays can have any type. For example:

```
String[] names; double[] vector; int[]
colors_list; char[] commands; boolean[] mask;
```

Instantiating Arrays: 2 Ways

- Way #1:

```
String[] s = new String [3];
s[0] = "You see an alligator.  What should
you do?  [Run] [Hide]";
s[1] = "Awww, you were eaten.  =(";
s[2] = "You escaped!";
```

Instantiating Arrays: 2 Ways

- Way #1:

```
String[] s = new String [3];
s[0] = "You see an alligator.  What should
you do?  [Run] [Hide]";
s[1] = "Awww, you were eaten.  =(";
s[2] = "You escaped!";
```

- Way #2:

```
int[] colors_list = {20, 40, 567433};
```

- Similar to Way #1, Way #2 creates a new array that can hold 3 elements.

Instantiating Arrays: 2 Ways

- Way #1:

```
String[] s = new String [3];
s[0] = "You see an alligator.  What should
you do?  [Run] [Hide]";
s[1] = "Awww, you were eaten.  =(";
s[2] = "You escaped!";
```

- Way #2:

```
int[] colors_list = {20, 40, 567433};
```

- Similar to Way #1, Way #2 creates a new array that can hold 3 elements.
- Arrays are indexed by integers, thus the following is true:

```
colors_list[1] == 40
```

Some Common Errors

`int array[x]`

- `x <= -1`

Some Common Errors

`int array[x]`

- `x <= -1`
- `x >= array.length`

Some Common Errors

`int array[x]`

- `x <= -1`
- `x >= array.length`
- `array = 5 \\This will cause an error`

Some Common Errors

int array[x]

- x <= -1
- x >= array.length
- array = 5 \\This will cause an error
- array[3] = 5 \\This is right

Some Common Errors

`int array[x]`

- `x <= -1`
- `x >= array.length`
- `array = 5 \\This will cause an error`
- `array[3] = 5 \\This is right`
- `array[x].length \\This will cause an error`

Some Common Errors

```
int array[x]
```

- `x <= -1`
- `x >= array.length`
- `array = 5 \\This will cause an error`
- `array[3] = 5 \\This is right`
- `array[x].length \\This will cause an error`
- `array.length \\This is right`

2 Dimensional Arrays

- Arrays can have more than one dimension. For example:

```
char[][] map = new map[20][20]
```

- This create a 20 × 20 array of characters.
- We can instantiate the array the same way as a 1D array. For example:

```
map[x][y] = 'S';
map =
{{'a','b','c'},{'d','e','f'},{'g','h','i','j'}};
```

- To get the number of variables in an array use `<arrayName>.length`

```
map.length == 3 && map[2].length == 4
```

Example

```
public class BattleShip{
   public static void main(String[] args){
   char[][] map = {{' ',' ',' ',' '},
                   {' ',' ','X',' '},
                   {' ',' ','X',' '},
                   {' ',' ','X',' '}};
```

Try printing out different squares in the array.

Complete the Code

```
__1__ __2__ TaxCalculator{
    __3__ __4__ __5__(__6__[] __7__){
        if (__8__.length __9__ 1) __10__
            __11__ tax = 0.075;
            double total = __12__.parseDouble(__13__);
            total __14__ 1.0 + tax;
            __15__.println("With a rate of " + tax
          + ", the total price is " + total + ".");
        }
    }
}
```

MiniQuiz: Find all the Errors

```
Public class _RollD20{
    Public statc void main(string[] args)
        int roll = Math.random;
        int d20 = Integer.parseInt(roll * 20)
        system.out.PrintLn('Your roll was ' +
           d + 20);
    }
```

Using a For Loop

- Now that we know about arrays, we can now use for loops to access many variables at once.
- For loops look like this:

```
for(int i = 0; i < 5; i++)
```

Using a For Loop

- Now that we know about arrays, we can now use for loops to access many variables at once.
- For loops look like this:

```
for(int i = 0; i < 5; i++)
```

- We can abstract for loops as:

```
for(<instantiation>; <boolean operation>;
        <arithmetic operation>)
```

Using a For Loop

- Now that we know about arrays, we can now use for loops to access many variables at once.
- For loops look like this:

```
for(int i = 0; i < 5; i++)
```

- We can abstract for loops as:

```
for(<instantiation>; <boolean operation>;
        <arithmetic operation>)
```

- Both the `<instantiation>` and the `<arithmetic operation>` are optional.

What does it do?

- Works like an if statement. If the `<boolean operation>` is true then it runs the next line or block of code.

What does it do?

- Works like an if statement. If the `<boolean operation>` is true then it runs the next line or block of code.
- Unlike an if statement the for loop keeps running the loop from top to bottom until the `<boolean operation>` is false.

What does it do?

- Works like an if statement. If the `<boolean operation>` is true then it runs the next line or block of code.
- Unlike an if statement the for loop keeps running the loop from top to bottom until the `<boolean operation>` is false.
- Before the loop begins `<instantiation>` is executed.

What does it do?

- Works like an if statement. If the `<boolean operation>` is true then it runs the next line or block of code.
- Unlike an if statement the for loop keeps running the loop from top to bottom until the `<boolean operation>` is false.
- Before the loop begins `<instantiation>` is executed.
- After each *iteration* of the for loop, the `<arithmetic operation>` is executed;

Printing a Map

Using the code from before, print out the array.

```
for(int y = 0; y < map.length; y++){
  for(int x = 0; x < map[y].length; x++){
     System.out.print(map[y][x]);
   }
   System.out.println();
}
```

Other Ways to Use a For Loop

- For loops are quite versatile:
- `for(int i = 0; i <5; i++)`

Other Ways to Use a For Loop

- For loops are quite versatile:
- `for(int i = 0; i <5; i++)`
- `for(String s = "a"; !s.equals("aaaaa"); s+="a"){...}`

Other Ways to Use a For Loop

- For loops are quite versatile:
- `for(int i = 0; i <5; i++)`
- `for(String s = "a"; !s.equals("aaaaa"); s+="a"){...}`
- `double d = 0.0; for(d = 1.0; d <5.0; d*=1.2);`

Other Ways to Use a For Loop

- For loops are quite versatile:
- `for(int i = 0; i <5; i++)`
- `for(String s = "a"; !s.equals("aaaaa"); s+="a"){...}`
- `double d = 0.0; for(d = 1.0; d <5.0; d*=1.2);`
- `boolean b = true for(; b; b=!b){...}`

Other Ways to Use a For Loop

- For loops are quite versatile:
- `for(int i = 0; i <5; i++)`
- `for(String s = "a"; !s.equals("aaaaa"); s+="a"){...}`
- `double d = 0.0; for(d = 1.0; d <5.0; d*=1.2);`
- `boolean b = true for(; b; b=!b){...}`
- `for(int x :  colors_list){...}`

Other Ways to Use a For Loop

- For loops are quite versatile:
- `for(int i = 0; i <5; i++)`
- `for(String s = "a"; !s.equals("aaaaa"); s+="a"){...}`
- `double d = 0.0; for(d = 1.0; d <5.0; d*=1.2);`
- `boolean b = true for(; b; b=!b){...}`
- `for(int x : colors_list){...}`
- `for( ; ; ){...} \\runs infinitely!!`

Other Ways to Use a For Loop

- For loops are quite versatile:
- `for(int i = 0; i <5; i++)`
- `for(String s = "a"; !s.equals("aaaaa"); s+="a"){...}`
- `double d = 0.0; for(d = 1.0; d <5.0; d*=1.2);`
- `boolean b = true for(; b; b=!b){...}`
- `for(int x : colors_list){...}`
- `for( ; ; ){...} \\runs infinitely!!`

BattleShip

Extend the Battleship code such that it works like the game Battleship. Have the computer randomly hide its ships using. Here's a hint:

```
int y = (int)(Math.random() * map.length);
```

Each turn the player should put in an input, which updates the map with either `'H'` for hit or `'M'` for miss.

BattleShip

Extend the Battleship code such that it works like the game Battleship. Have the computer randomly hide its ships using. Here's a hint:

```
int y = (int)(Math.random() * map.length);
```

Each turn the player should put in an input, which updates the map with either `'H'` for hit or `'M'` for miss.

Optionally, extend the code so that you can place your own ships, Give the computer an AI for playing using `Math.random()`.

Spot the Syntax Errors

```
int array = new array[-1];
for(int y = -1; y <= array.length; y <= 1){
    array[y] = array[y + 1];
    array = null;
}
```

Spot the Logic Errors

- We want the old array to hold the contents of the array.
- We want the new array to contain {9, 7, 5, 3, 6}

```
int oldarray = array;
int[] array = {5, 4, 3, 2, 1};
array = oldarray;
for(int y = 0; y < array.length; y ++){
    array[y] += array[(y + 1) % 4];
}
```

While Loops

- While loops work just like for loops, except they only use booleans:

```
while(<boolean operation>)
```

- Example:

```
String s ="";
while(!s.equals("Yes")){
    System.out.println("Are you sure?")
    s = input.nextLine();
}
```

Do - While Loops

- Do-while loops work the same way as while loops except *they always execute the following block of code at least once.*
- Example:

```
String s = input.nextLine();
do {
    System.out.println("Are you sure?")
    s = input.nextLine();
}while(!s.equals("Yes"));
```

- Challenge: change your Battleship code to use only while or do-while loops.

Equivalence between For and While Loops

```
for(<instantiation>; <boolean operation>;
     <arithmetic operation>){
...
}
```

$$\equiv$$

```
<instantiation>;
while(<boolean operation>){
...
<arithmetic operation>
}
```

Switch-Case

- Often in programming we need to check several different cases. For example:

```
int i = 5;
if (i == 0){
...
} else if (i == 1){
...
} else if (i == 2){
...
} else ....
```

- This takes a long time. Instead, we can use Switch-Case statements.

Quick Experiment!

```
int i = ___;
switch(i){
   case 0:
   System.out.println("0!");   break;
   case 1:
   System.out.println("1!");   break;
   case 2:
   System.out.println("2!");
   ...
```

Quick Experiment!

```
    case 3:
    System.out.println("3!");
    case 4:
    System.out.println("4!");   break;
    default:
    System.out.println("Other");
}
```

Why use Switch-Case?

- Switch case simplifies code, and gets rid of if-else chains.

Why use Switch-Case?

- Switch case simplifies code, and gets rid of if-else chains.
- It is much faster!

Convert the experiment code to a chain of if-else statements. Use `System.nanoTime()` to measure how long the code takes. Example:

Why use Switch-Case?

- Switch case simplifies code, and gets rid of if-else chains.
- It is much faster!

Convert the experiment code to a chain of if-else statements. Use `System.nanoTime()` to measure how long the code takes. Example:

```
long timeBefore = System.nanoTime();
\\ Your code here.
long timeAfter = System.nanoTime();
long totalTime = timeAfter - timeBefore;
```

Why use Switch-Case?

- Switch case simplifies code, and gets rid of if-else chains.
- It is much faster!

  Convert the experiment code to a chain of if-else statements. Use `System.nanoTime()` to measure how long the code takes. Example:

```
long timeBefore = System.nanoTime();
\\ Your code here.
long timeAfter = System.nanoTime();
long totalTime = timeAfter - timeBefore;
```

- Why use if-statements?

Why use Switch-Case?

- Switch case simplifies code, and gets rid of if-else chains.
- It is much faster!

  Convert the experiment code to a chain of if-else statements. Use `System.nanoTime()` to measure how long the code takes. Example:

  ```
  long timeBefore = System.nanoTime();
  \\ Your code here.
  long timeAfter = System.nanoTime();
  long totalTime = timeAfter - timeBefore;
  ```
- Why use if-statements? *You need if statements for any non "discrete" values, like doubles or strings.*

Pop Quiz!

What is the difference between Declaration and Instantiation?

Pop Quiz!

What is the difference between Declaration and Instantiation? Can an array be declared and have all of its variables instantiated in one line?

Pop Quiz!

What is the difference between Declaration and Instantiation?
Can an array be declared and have all of its variables instantiated in one line?
What does `System.nanoTime()` do?

Pop Quiz!

What is the difference between Declaration and Instantiation?
Can an array be declared and have all of its variables instantiated in one line?
What does `System.nanoTime()` do?
How do you make an infinite for loop?

Pop Quiz!

What is the difference between Declaration and Instantiation?
Can an array be declared and have all of its variables instantiated in one line?
What does `System.nanoTime()` do?
How do you make an infinite for loop?
Are while loops and for loops equivalent?

Pop Quiz!

What is the difference between Declaration and Instantiation?
Can an array be declared and have all of its variables instantiated in one line?
What does `System.nanoTime()` do?
How do you make an infinite for loop?
Are while loops and for loops equivalent?
Give an example of a scenario where we can't use switch-case statements.

Implement a 2D map based RPG

From now until Lunch, try to implement a 2D RPG with a Player, 'P', and a monster 'M'. The player should battle the monster if the player is on the same square.

WarmUp

```
import java.util.Scanner;

public class quizGame{
   static String[] questions = {"Pick a number (1-4)",
   "Pick a color (1. Red, 2. Blue, 3. Green, 4. Purple)",
   "Pick an animal (1. Panda, 2. Squirrel, 3. Whale, 4. Jigglypuff) "};

   public static void main(String[] args){
      int i = 0;
      Scanner input = new Scanner(System.in);
      for(String s : questions){
         System.out.println(s);
         i = (i + input.nextInt())%4;
      }
```

WarmUp

```
switch (i){
   case 0 :
   System.out.println("You will have a super lucky day!");
   break;
   case 1 :
   System.out.println("You will learn a lot today!");
   break;
   case 2 :System.out.println("You will have a bad day today.");
   break;
   case 3 :System.out.println("Today is a good day to make friends");
   break;
   default:
   System.out.println("Error!");
}
input.close();
}
}
```

A simple method

- Believe it or not, we know enough syntax to make any program conceivable (though we do not know other things like how to interface to the screen).

A simple method

- Believe it or not, we know enough syntax to make any program conceivable (though we do not know other things like how to interface to the screen).
- However, if we put everything in the main method, our code will be extremely long and unreadable!
- We need methods:

```
public static <returnType>
<methodName>(<OptionalArguments>){ ...  }
```

- A *static* method can take in *arguments*, return a value, and change the state of other static variables. Lets try it:

```
public static int RollXd20(int number){
   return (int)(number*20*math.Random()) +1;
}
```

Static vs. Non-Static

- For the time being we are going to keep everything *public*.
- However, we can start using *non-static*, instance variables and methods.

Static vs. Non-Static

- For the time being we are going to keep everything *public*.
- However, we can start using *non-static*, instance variables and methods.
- So far we've used a few classes that require *instantiation* (e.g. Scanner, arrays)

Static vs. Non-Static

- For the time being we are going to keep everything *public*.
- However, we can start using *non-static*, instance variables and methods.
- So far we've used a few classes that require *instantiation* (e.g. Scanner, arrays)
- *Static* methods and variables do not require instantiating a class.
- Non static methods do require instantiating an object, and each instance gets its own copy of all the non-static variables.

Creating an Object

```
public class Person{
    static int population = 0;
    int counter = 0;
    int ssn;

    public Person(int ssn){
       this.ssn = ssn;
       population++;
       counter ++;
    }

    public in getSSN(){
       return ssn;
    }
}
```

Instantiating an Object

```
public class PersonTest{
    public static void main(String[] args){
        Person p = new person();
        p.getSSN();
    }
}
```

Case study: Objects with a Static Variable

Create a static method in the Person class that returns the population variable. Next in the PersonTest class, create more than one person using a loop. How does the population change? Finally, do the same for the counter variable. Does it change?

Refactor your 2D RPG into methods

Your RPG from before lunch can be augmented with methods. For example, you can put all of the logic for the monster in a monster method. Try to put all of the logic for the monster in its own method, then recompile your program and make sure it works.

Getting organized

As you get more classes, you may want to start sorting them into different packages. Creating a package is easy. In Eclipse click File→New→Package. Name your package and click Finish. Don't forget, it you're using code that's in a different package, you need to use *import*.

Extra Challenge

Move your methods into different classes and or different packages. For example it might be nice to have a dice package that calculates random dice numbers for 4d20, d6, etc. After this, if your RPG is complete enough show it to me or a friend!

A Weird Way to Loop

- Loops let us repeat the same code more than once, but they have one limitation: complexity.

A Weird Way to Loop

- Loops let us repeat the same code more than once, but they have one limitation: complexity.
- Sometimes code written using loops can be verbose and confusing, and for this reason programmers sometimes opt to use recursion.
- What is a recursive function?

A Weird Way to Loop

- Loops let us repeat the same code more than once, but they have one limitation: complexity.
- Sometimes code written using loops can be verbose and confusing, and for this reason programmers sometimes opt to use recursion.
- What is a recursive function?
- Definition: $f(x) = g(f(x-1), f(x-2), ...)$. In other words if we have some output f_{n+1} it depends on its previous output(s) f_n.

A Weird Way to Loop

- Loops let us repeat the same code more than once, but they have one limitation: complexity.
- Sometimes code written using loops can be verbose and confusing, and for this reason programmers sometimes opt to use recursion.
- What is a recursive function?
- Definition: $f(x) = g(f(x-1), f(x-2), ...)$. In other words if we have some output f_{n+1} it depends on its previous output(s) f_n.
- Example: $f_{n+1} = !f_{n+1}$, where for all n, f_n is either `true` or `false`.

A Weird Way to Loop

- Loops let us repeat the same code more than once, but they have one limitation: complexity.
- Sometimes code written using loops can be verbose and confusing, and for this reason programmers sometimes opt to use recursion.
- What is a recursive function?
- Definition: $f(x) = g(f(x-1), f(x-2), \ldots)$. In other words if we have some output f_{n+1} it depends on its previous output(s) f_n.
- Example: $f_{n+1} = !f_{n+1}$, where for all n, f_n is either `true` or `false`.
- This gives us the sequence `true, false, true, false, true`...
- What does this look like in code?

Recursion

```
class factorial{
   public static void main(String[] args){
      int i = 0;
      if (args.length > 0){
         i = Integer.parseInt(args[0]);
      }
      System.out.println(i + "! = " + factorial(i));
   }
```

Recursion

```
    public static int factorial(int n){
       if(n > 1){
          return n * factorial(n-1);
       }
       else if(n >= 0){
          return 1;
       }
       return -1;
    }
}
```

Questions

What number sequence does this output?

$f_0, f_1, f_2, f_3 \ldots ?$

Questions

What number sequence does this output?

$f_0, f_1, f_2, f_3 \ldots$?

Challenge: Which is faster, recursion or looping? Can you use `System.nanoTime()` to find out?

2D Array based Collision Algorithm

Download the resources file on my website into your MyProjects folder, and then extract its contents. Import the resources as a project into Eclipse. Click File → Import → General → Existing Project from Workspace → Next → Browse. Then select the Resources folder. Finally click Select All → Finish.

The interesting method is moveCell() in SimpleSolid.java

Some Big If Statements

```
if (temp_x >= 0 && temp_x < w.map.length
      && temp_y >= 0 &&
      temp_y < w.map[0].length) {
   SimpleSolid s = w.map[temp_x][temp_y];
   if (s != null) {
      s.collision(this);
      collision(s);
   }
```

Some Recursive Problems

- $f_{n+1} = f_n + 1$ for $f_0 >= 0$
- $f_{n+1} = f_n + 2$ for $f_0 >= 1$
- $f_{n+1} = f_n^{*}2$ for $f_0 = 1$
- $f_{n+1} = f_n / f_{n-1}$ for $f_1 = 2, f_0 = 1$

Some Recursive Problems

- $f_{n+1} = f_n + 1$ for $f_0 >= 0$
- $f_{n+1} = f_n + 2$ for $f_0 >= 1$
- $f_{n+1} = f_n^* 2$ for $f_0 = 1$
- $f_{n+1} = f_n / f_{n-1}$ for $f_1 = 2, f_0 = 1$
- SUPER BONUS: Try to come up with a recursive function that outputs a random number sequence.

Some Recursive Problems

- $f_{n+1} = f_n + 1$ for $f_0 >= 0$
- $f_{n+1} = f_n + 2$ for $f_0 >= 1$
- $f_{n+1} = f_n^* 2$ for $f_0 = 1$
- $f_{n+1} = f_n / f_{n-1}$ for $f_1 = 2, f_0 = 1$
- SUPER BONUS: Try to come up with a recursive function that outputs a random number sequence.
- SUPER SUPER BONUSE: Create the fill-in algorithm using recursion.