Lesson 29: Quiz #3 Prep & `StarPrinter` (W08D4)

Balboa High School

Michael Ferraro

October 8, 2015

Do Now

Today's lesson is split into two parts:

- *Part I:* Self-guided Quiz #3 practice
- *Part II:* `StarPrinter`, a Java class you'll create with a partner to help you complete part of PS #4b, §8

At this time, begin moving through the subsequent slides. I'll be around to answer your questions. Once you reach *Part II*, find a partner — who is also done with *Part I* — with whom to develop `StarPrinter`.

Aim

Students will review and practice for Quiz #3 (next class!) and work on `StarPrinter`.

Part I

Self-Guided Quiz #3 Practice — record all answers on a sheet of paper.

`ArrayList` Concepts

- We covered the most basic `ArrayList` concepts
- Know how to...
  - create a new `ArrayList` object
  - add an element via the `add()` method
  - retrieve a specific element via `get()` & `remove()`[1]
  - get the # of elements via `size()`
- Understand how `ArrayList` elements are *indexed*
  - index 0 is the 1st element
  - index 1 is the 2nd element
  - index $(n-1)$ is the n^{th} (i.e., last) element

[1] `remove()` has the effect of **both** removing an element and retrieving it!

ArrayList Practice

```
ArrayList<Integer> aList = new ArrayList<Integer>();
aList.add(-17); aList.add(5); aList.add(0); aList.add(23);

int a = aList.get(1);
int b = aList.get(4);
```

AL1) What is the index of the element containing `0`?

AL2) What would a call to `aList.size()` return?

AL3) What will be the value of `a`?

AL4) What will be the value of `b`?

Special Operators

You should be familiar with these:

`i++;`	**incrementer**, increases value by 1
`i--;`	**decrementer**, decreases value by 1
`i += 5;`	shorthand for `i = i + 5;`
`i -= 3;`	shorthand for `i = i - 3;`
`i *= -2;`	shorthand for `i = i * -2;`
`i /= 9;`	shorthand for `i = i / 9;`

Iteration

- We used `while()` to perform actions more than once, looping — or *iterating*
- Non-∞ loops must have...
  - a **stopping condition** — when the expression in `while`'s "`()`" is no longer true, and
  - a **loop variant** — that which changes with each iteration, getting closer to stopping condition

Iteration Practice #1

```
int a = 0, x = 1;

while ( a < 5 ) {
    x *= a - 1;
    a++;
}
```

ITN1) How many iterations will this loop complete?

ITN2) What's the final value of `x`?

Iteration Practice #2

```
int c = 2;

while ( c >= 0 ) {
    c--;
    c *= 2;
}
```

ITN3) How many iterations will this loop complete?

ITN4) What's the final value of c?

Iteration Practice #3

ITN5) Write a Java method that takes an integer, `n`, and returns its factorial. The method must not call itself (i.e., cannot be recursive).

Precondition: $n > 0$.

ITN6) Write a Java method that takes an integer, `n`, and returns the *sum of cubes* from 1 to `n`. For example, if the method is passed the value 3, it must return the value of $1^3 + 2^3 + 3^3$, or 36.

Precondition: $n > 0$.

Recursion

- When a problem can be written in terms of itself, it can be implemented recursively
- Two characteristics of all recursive procedures are
  - call to itself, and
  - base case to stop the recursion

Recursion Practice

```
public static int playTime(int p) {
    if ( p > 2 ) {
        return 4;
    }
    return p + playTime( p - 1 );
}

public static void main(String[] args) {
    int m = playTime(4);
    int n = playTime(-1);
}
```

REC1) What's the final value of `m`? Show the recursive calls using the technique you were shown in class — or a suitable equivalent!

REC2) Do the same for `n`.

Search Techniques

- Two Methods
  - *Sequential Search* — searching through every element until you find what you're looking for; akin to flipping over playing cards from left to right, looking for a specific card
  - *Binary Search* — e.g., guessing a number by always comparing midpoint to what you're searching for, requires data is *ordered*
- Performance characteristics
  - Best Case — least "guesses" required when...
  - Average Case — over many trials, ...
  - Worst Case — most "guesses" required when...

Make sure you can. . .

You should be able to implement the following algorithms on paper during the test. Make sure you understand how each one works (i.e., can you draw a quick flowchart?), and then how you might write Java code to carry out the process.

- *Sum of Squares*
- *Euclid's GCF*
- *Factorial*

Each of those algorithms can be implemented reasonably using iterative or recursive procedures. So long as you can write a working method using one of those types, you can get full credit!

Grade Yourself

```
AL1.  2
AL2.  4
AL3.  5
AL4.  no value, IndexOutOfBounds Exception!

ITN1.  5
ITN2.  0
ITN3.  infinite # of iterations
ITN4.  alternates between 1 & 2
```

Grade Yourself

```
//ITN5 -- counting down on n:

public static int factorial(int n) {
    int result = 1;

    while ( n > 0 ) {
        result = result * n;
        n--;
    }

    return result;
}

//alternatively, you may have introduced
//an extra variable and counted up
```

Grade Yourself

```
//ITN6 -- counting up

public static int sumOfCubes(int n) {
    int sum = 0, a = 1;

    while ( a <= n ) {
        sum = sum + a*a*a;
        a++;
    }

    return sum;
}
```

Grade Yourself

```
//ITN6 -- counting down

public static int sumOfCubes(int n) {
    int sum = 0;

    while ( n > 0 ) {
        sum = sum + n*n*n;
        n--;
    }

    return sum;
}
```

Grade Yourself

```
REC1.  4 -- base case reached immediately


REC2.  infinite recursion -- no result returned,
       base case never reached!

       (in reality,
       java.lang.StackOverError thrown)
```

Part II

`StarPrinter` — find a partner!

StarPrinter: printStars()

- In Eclipse, you should already have a project called `ps4b`; add a new class to that project called `StarPrinter`.
- Add this method:

```
public static void printStars(int n) {

    while(n > 0) {
        System.out.print("*");
        n--;
    }

    System.out.print("\n");
}
```

StarPrinter: printStars()

- In Eclipse, you should already have a project called `ps4b`; add a new class to that project called `StarPrinter`.
- To test the method, add a `main()` with these lines of code:

```
printStars(1);    //should print  |*
printStars(0);    //                |
printStars(1);    //                |*
printStars(2);    //                |**
printStars(3);    //                |***
```

StarPrinter: printTriangle()

- Now you're ready to work on PS #4b, §8.2, #1
- Add a method called `printTriangle()`, which is to be called with an integer argument.
  - *precondition:* the argument will be ≥ 1
  - `printTriangle()` must call `printStars()`
  - the table on the next slide shows the printed output for specific calls to `printTriangle()`

printTriangle() Output

printTriangle(1)	printTriangle(2)
*	* **

printTriangle(3)	printTriangle(5)
* ** ***	* ** *** **** *****

HW

- Study for Quiz #3!
  - flip through lesson slides re: algorithms (recursive & iterative)
  - re-attempt some of the hands-on exercises from class and the problem sets
- Continue working on PS #4b. You'll have some time after Quiz #3 is over to work on it, and it'll be due by the **beginning of the following class.**