Darrel Chang
Lan Yang
CS 4080
11/13/24

Homework 4

1. (4 points) Default parameter values.
a. Use your own words to describe what is meant by "default parameter values" (or, simply default parameters)?

Default parameters are function parameters that are used if no parameter is passed in, which allow the function to still be called if parameters are omitted from the call.

b. Use pseudo code write a function definition and two function calls to illustrate the usage of default value (one call uses default value, while the other call doesn't use default value).

```
function greet(name = "Guest):
   print("Hello " + name + "! ")

greet() // Prints Hello Guest!

greet("Alice") // Prints Hello Alice!
```

c. What is/are the advantage(s) of using default value?

Using a default value allows a function to still be used in the use case where a parameter is passed in. This allows for better code writability without the need to write multiple functions to cover different amounts of passed parameters. It also improves the readability of the code, because its default function is better stated in the function definition.

d. Does Java support default parameter values? Why or why not?

Java does not support default parameter values. This limitation is an intentional language design because Java emphasizes strict typing and method signatures for function calls.

2. (4 points) Consider the following program written in C syntax with parameter passing method indicated in each of the following questions:

```
void swap(int a, int b) { int temp; temp = a; a = b; b = temp;}
void main() {
int seed = 2, list[5] = {10, 12, 14, 16, 18};
swap(seed, list[2]);
swap(list[0], list[2]);
seed = 3;
swap(seed, list[seed]); //now, seed = ? list = ?
}
```

(a) under pass by value, what are the values of the variables seed and list at the end?

Under pass by value the parameters that are passed in are not changed because a copy of it is swapped in the function. This means seed and list remain the same before and after the function call. seed == 3 because it was assigned to 3 before the last function call. list remains the same as its initial state == {10, 12, 14, 16, 18}

(b) under pass by reference, what are all of the values of seed and list at the end?

Under pass by reference the function will swap the parameters passed in.
Initial:
seed = 2, list = {10,12,14,16,18}

swap(seed, list[2])
seed = 14 list = {10,12,2,16,18}
swap(list[0], list[2])
seed = 14 list = {2,12,10,16,18}
seed = 3
swap(seed, list[seed])

Final:
seed = 16, list = {2,12,10,3,18}

3. (3 points) For the following code segment, assume Java's parameter passing method is used, will the values of first and second be swapped? Why or why not? Explain briefly.

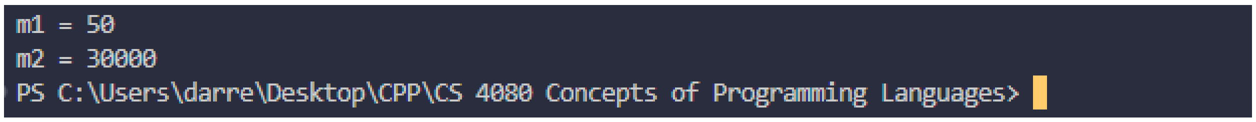
```
public class myInteger {
        public int value;
        myInteger (int i) {value = i;}
}
public static void swap (myInteger a, myInteger b) {
        myInteger temp = a;
        a = b;
        b = temp;
        return;
}
int first = 7, second = 5;
myInteger one = new myInteger (first);
myInteger two = new myInteger (second);
swap(one, two);
first = one.value;
second = two.value;
//will first = 5 and second = 7?
```

With Java's parameter passing method, the values of first and second will not be swapped. This is because the references of a and b are passed by value, so the original references are never changed.

4.
(4 points) The following is C# like code segment for demonstrating variable length arguments, please rewrite it to Java in the form of a complete program. Run the Java code and show results (you may use screenshot to show code and results.)

```
void Main() {
int m1 = Multiply(5, 10);
int m2 = Multiply(50, 20, 30);
//now display m1 and m2 values.
}
int Multiply(params int[] b) {
int mul =1;
foreach (int a in b) {
mul = mul*a;
}
return mul;
}
}
```

```
public class main {
	public static void main(String[] args) {

		int m1 = multiply(5, 10);
		int m2 = multiply(50, 20, 30);
		System.out.println("m1 = " + m1);
		System.out.println("m2 = " + m2);
	}

	public int Multiply(int… numbers) {
		int res = 1;
		for (int num : numbers) {
			res *= num;
		}
		return res;
	}
}
```

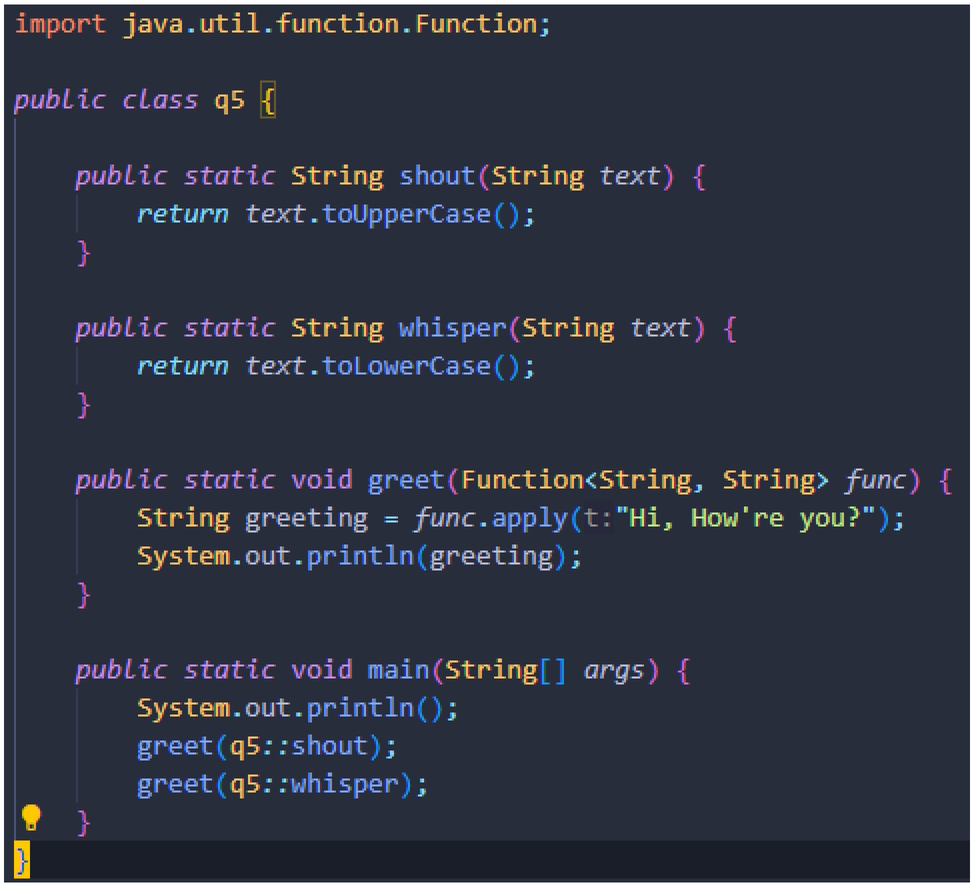


```
m1 = 50
m2 = 30000
PS C:\Users\darre\Desktop\CPP\CS 4080 Concepts of Programming Languages>
```

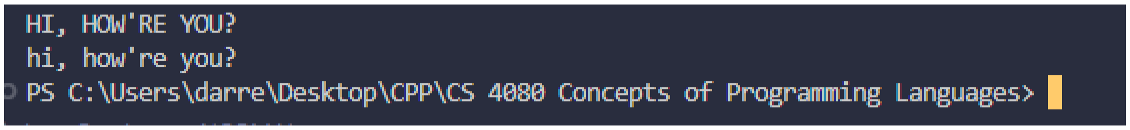
5.
5 points) The following is Python like code segment that demonstrate passing function/subprogram as argument.

```
def shout(text):
return text.upper()
def whisper(text):
return text.lower()
def greet(func):
# storing the function in a variable
greeting = func("Hi, How’re you? ")
print(greeting)
greet(shout) #output: HI, HOW’RE YOU?
greet(whisper) #output: hi, how’re you?
```

(a) Please rewrite it to Java and run the Java code. In the rewritten code, should keep the original design idea, i.e. shout(), whisper(), and greet() functions should be properly defined in the same prototypes as in the original code (e.g. shout() takes a text as parameter) and two calls to the greet() function in main with a function as parameter. Attach code and result as answer to this question.



```java
import java.util.function.Function;

public class q5 {

    public static String shout(String text) {
        return text.toUpperCase();
    }

    public static String whisper(String text) {
        return text.toLowerCase();
    }

    public static void greet(Function<String, String> func) {
        String greeting = func.apply(t:"Hi, How're you?");
        System.out.println(greeting);
    }

    public static void main(String[] args) {
        System.out.println();
        greet(q5::shout);
        greet(q5::whisper);
    }
}
```



```
HI, HOW'RE YOU?
hi, how're you?
PS C:\Users\darre\Desktop\CPP\CS 4080 Concepts of Programming Languages>
```

(b) Compare the readability and writability of the two codes (i.e. the given Python code vs. your Java code.)

I think python wins in this case in both readability and writability due to its simplistic nature. Python can just pass in functions as parameters, while Java requires boilerplate code such as defining the functional interface Function and importing. Python has the disadvantage of being less reliable because of its implicit typing, while Java's string explicit typing makes the code more robust.