Exam 2 Review

Colorado State University

Department of Computer Science

Slides Originally Created by Albert Lionelle (Albert.Lionelle@colostate.edu),
updated by Marcia Moraes (marcia.moraes@colostate.edu)

Announcements

https://www.wilkinsontriad.com/dont-give-up-you-are-almost-there

TODO Reminders:

- Canvas Exam 2 – during your Lab time Thursday March 9th
- All Lab retakes – due to Friday March 10th
- Coding Exam 2 – due to Friday March 10th – available until March 12
- Reflections: Self-reflection and RPA Reflections (Part1-3) – due to Friday March 10th – available until March 12
- No Help Desk on Saturday March 11th and Sunday 12th, neither next week – Spring Break

Keep practicing your RPAs in a spaced and mixed manner ☺

While and Do…While Loops

- **while** loops - checks condition before running
- **do while** loops - runs *once always*, and then checks the condition.

```
int val = 10;

while(val < 10) {
    System.out.println("While: Does this print?");
}

do {
    System.out.println("Do-While: Does this print?");
}while(val < 10);
```

What is the output?

It is not going to enter the while

Will print one time

this

`this` literally means "this instance / object", so used when you want to reference *instance* variables.

Is the setID method of Student class correctly implemented?

```
public class Student {
    private int id;
    public Student(){
        id = 5;
    }
    public void setID(int id){
        id = id;
    }
    public String toString(){
        return String.format("ID: %d", id);
    }
}
```

What is going to be printed?

```
public class AppStudent {
    public static void main(String args[]){
        Student stuie = new Student();
        stuie.setID(10);
        System.out.println(stuie);
    }
}
```

```
ID: 5
```

How to correct it?

this

`this` literally means "this instance / object", so used when you want to reference *instance* variables.

What is going to be printed now?

```
public class Student {
   private int id;
   public Student(){
      id = 5;
   }
   public void setID(int id){
      this.id = id;
   }
   public String toString(){
      return String.format("ID: %d", id);
   }
}
```

```
public class AppStudent {
   public static void main(String args[]){
      Student stuie = new Student();
      stuie.setID(10);
      System.out.println(stuie);
   }
}
```

```
ID: 10
```

this

```
Student stuie = new Student();
```

We see the *new* keyword, which means create a new table:

Student@x131

variable	value
id	5

The Stack (initial memory location)

variable	value
stuie	Student@x131

this

Now, we call the following line of code

```
stuie.setID(10);
```

Which means, we are *inside* of stuie, giving us access to the following variables:

.setID(10)

variable	value
id	10
this.id	5

Overloaded Methods

What is an overloaded **method**?

Why are they useful?

Overloaded Constructor and this

- Constructors are specialized methods whose purpose is to 'build' the object
  - They can only be called via the 'new' keyword
- Overloading a Constructor is just like overloading a method!
  - It creates optional ways to create objects.

Overloaded Constructor and this

```
public class Student {
   private int id;
   private String name;
   public Student(){
      this(5, null);
   }
   public Student(int id){
      this(id, null);
   }
   public Student(int id, String name){
      setID(id);
      this.name = name;
   }
   public void setID(int id){
      this.id = id;
   }
   public String toString(){
      return String.format("ID: %d Name: %s", id, name);
   }
}
```

What is going to be printed?

```
public class AppStudent {
   public static void main(String args[]){
      Student stuie = new Student();
      Student stuie2 = new Student(10);
      Student stuie3 = new Student(20,"Alice");
      System.out.println(stuie);
      System.out.println(stuie2);
      System.out.println(stuie3);
   }
}
```

```
ID: 5 Name: null
ID: 10 Name: null
ID: 20 Name: Alice
```

Substring and IndexOf

- substring(start, end) - takes the substring from index start (inclusive) to index end (exclusive)
  - Since it takes int values, it pairs well with indexOf.
- indexOf(string or char) - gives you the location where that item **first** shows up or -1 if it isn't in the string
- indexOf(string or char, int) - gives you the location of the item the first time it shows up after (including) the int location for starting
  - `indexOf(string or char)` is the equivalent of `indexOf(string or char, 0)`

Substring and IndexOf

So let's take the following code

```
String knocker = "tattarrattat";
String knock = knocker.substring(knocker.indexOf("r")+1,
                 knocker.indexOf("t", knocker.length()-1)+1);
```

Step 1: I would write out the string, and put the indices under it!

t	a	t	t	a	r	r	a	t	t	a	t
0	1	2	3	4	5	6	7	8	9	10	11

Substring and IndexOf

Step 2: read the code and take it in **smaller parts** (divide-conquer-glue)

- knocker.indexOf("r") + 1 - i know this starts at 0, goes to r, but then adds 1 to it
  - 5+1 = 6
  - `knocker.substring(6, ...)`
- knocker.length()-1
  - 12 - 1 = 11
  - `knocker.indexOf("t",11)+1)`
    - 11 + 1 = 12
    - yes, just a way to grab the last T
- `knocker.substring(6,12);`
- I then remind myself inclusive, exclusive

r	a	t	t	a	t
6	7	8	9	10	11

Substring and IndexOf

```
// your turn - at the table, work it out with the same steps above!

String plant = "kinnikinnick";
String p2 = plant.substring(plant.indexOf("k"),
            plant.indexOf("k", plant.indexOf("i"))+1);

System.out.println(p2);
```

```
kinnik
```

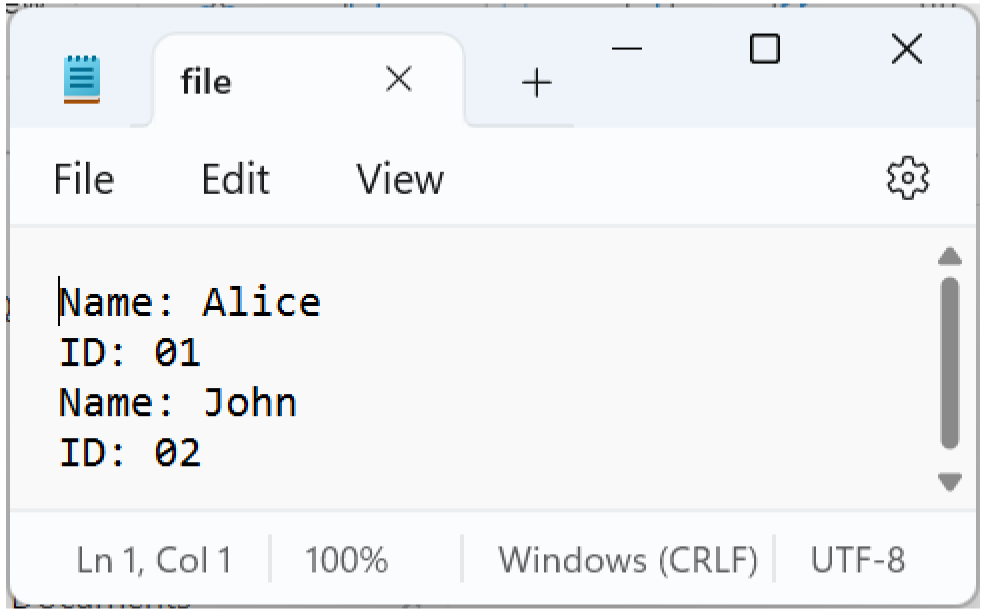
File Input

```
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Scanner;
public class FileReading {
   public static void main(String args[]){
      ArrayList<String> lst = new ArrayList<>();
      try{
         Scanner file = new Scanner(new File("file.txt"));
         while(                ){


         }
      }catch(                    ){
         System.out.println("File not found!");
      }
      for(String s: lst){
         System.out.println(s);
      }
   }
}
```

Fill in the blanks to make this code to work and have the following output, considering the file presented:

```
Name: Alice
ID: 01
Name: John
ID: 02
```



File Input

```
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Scanner;
public class FileReading {
   public static void main(String args[]){
      ArrayList<String> lst = new ArrayList<>();
      try{
         Scanner file = new Scanner(new File("file.txt"));
         while(file.hasNext()){
            String line = file.nextLine();
            lst.add(line);
         }
      }catch(IOException ex){
         System.out.println("File not found!");
      }
      for(String s: lst){
         System.out.println(s);
      }
   }
}
```

RPA Questions

1 1 point

Given the following code, answer the questions:

```
 int value = 0;
StringBuilder build = new StringBuilder();
do {
  build.append(value++);
} while (++value < 10);

System.out.println(build.toString()); // print 1

String hello = "What's----the---motto---with-you?";
while(hello.contains("--")) {
 hello = hello.replaceFirst("---", "-");
  hello = hello.replaceFirst("--", "-");
  System.out.println(hello);
}

System.out.println(hello); // print 2
```

What is printed in //print 1?

What is printed in //print 2?

RPA Questions

1 point

Given the following code, answer the questions:

```
 int value = 0;
StringBuilder build = new StringBuilder();
do {
  build.append(value++);
} while (++value < 10);

System.out.println(build.toString()); // print 1

String hello = "What's----the---motto---with-you?";
while(hello.contains("--")) {
 hello = hello.replaceFirst("---", "-");
  hello = hello.replaceFirst("--", "-");
  System.out.println(hello);
}

System.out.println(hello); // print 2
```

What is printed in //print 1?

```
02468
```

What is printed in //print 2?

```
What's-the-motto-with-you?
```

RPA Questions

Order the classes from parent to child (super to inheriting class)

Anyone (both classes that extend, and those that don't) can create a **GameObject** with MOVABLE set to true.

```
public class GameObject {
    public final int ID;
    public final boolean MOVABLE;

    public GameObject(int id) {
        this(id, false);
    }

    protected GameObject(int id, boolean canMove) {
        ID = id;
        MOVABLE = canMove;
  }
}
----
public class MobileObject extends GameObject {
    private int movement = 10;

    public MobileObject(int id) {
        super(id, true);
    }

    public int getMovement() {
        return movement;
    }

    public void setMovement(int movement) {
        this.movement = movement;
  }
}
```

```
public class NamedObject extends MobileObject {
    public final String name;
    private int level = 1;


    public NamedObject(int ID, String name) {
        super(ID);
        this.name = name;
    }


    public void setLevel(int level) {
        this.level = level;
    }


    public int getLevel() {
        return level;
    }


    public int getMovement() {
        return super.getMovement() * getLevel();
  }
}
```