ESE112
Java Programming:
- Primitive vs. References Type
- Stack vs. Heap
- Null Pointer Exception
- Keyword this
- Strings
- Has a Relationship

**Primitive vs. Reference Types**
- We've seen Java's 4 *primitive* types: int, double, boolean, char
- Types other than the primitive types are known as *reference* types
  - Used for objects
- Examples of reference variables:
  - `Student s1; Counter c1; String name;`
  - Note: String is an object not primitive type

**Stack vs. Heap**
- Stack is section of computer's memory used to store temporary information
  - E.g. Variable created inside a method
  - Information ceases to exist after method finishes execution
- Objects (i.e. object’s data) are stored in section of memory called heap
  - Information exists as long as the program is not finished

**How the Stack Works**

Assume that statements are with a method

<table>
<thead>
<tr>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int x = 0;</code></td>
</tr>
<tr>
<td><code>x = 5;</code></td>
</tr>
<tr>
<td><code>double min = 0.5;</code></td>
</tr>
<tr>
<td><code>boolean done = false;</code></td>
</tr>
</tbody>
</table>

Note: Variables are added in the order they are declared.
Reference Type

- The term *reference* is used because it refers to a memory location where the object lives
  - The variable of reference type is used to access the object

- The value of variable of reference type is either “null” or a “heap address”
  - *null* means currently not pointing at any location

Example:
```
Student s1; // s1 => null
s1 = new Students();
```
- *s1* now refers to a location in memory where the Student object that *s1* is pointing resides

Null Pointer Exception

- *null* is a legal value for any kind of object
  - i.e. Person p, Counter c; Player mario

- *null* can be assigned, tested, and printed

- But if you try to use a field or method of *null*, you get a *nullPointerException* i.e. you try to access some object that has not been created
  - E.g. Student s1;
    ```
    s1.getName();
    ```

Counter Class Example

```java
public class Counter{
    private int count;

    public Count(){
        count = 0;
    }
    public void incCount(){
        count++;
    }
}
```

How the Heap Works

<table>
<thead>
<tr>
<th>Assume statements are written within main()</th>
<th>Stack and Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>int x = 99;</td>
<td></td>
</tr>
<tr>
<td>Counter c1</td>
<td></td>
</tr>
<tr>
<td>c1 = new Counter();</td>
<td></td>
</tr>
<tr>
<td>c1.incCount();</td>
<td></td>
</tr>
<tr>
<td>c1.incCount();</td>
<td></td>
</tr>
<tr>
<td>Counter c2 = new Counter();</td>
<td></td>
</tr>
<tr>
<td>c2</td>
<td></td>
</tr>
</tbody>
</table>
Aliases

- Two or more references can point to the same object
  - These references are then known as aliases

Student s2 = new Student("Lisa", 5);
Student s3 = null;
Student s2 = s3; //s2 and s3 are now aliases
System.out.println(s2.getAge());
System.out.println(s3.getAge()); //returns same value as the above statement

Scope Issues with variables

public class Dot{
    private int x;
    private int y;
    public Dot(int x, int y){
        x = x;  // problem!!
        y = y;
    }
}

Solution

public class Dot{
    private int x;
    private int y;
    public Dot(int x, int y){
        this.x = x;  // fixed!!
        this.y = y;
    }
}

Keyword this

- this is a reference (a variable) to the current object
  - The object whose method or constructor is being called
  - The value of “this” is an object’s heap address
    - Can be passed as argument or input to a method
    - Can be returned as value
    - However it cannot be modified (i.e. it is final)
  - Used to Differentiate objects & avoid scope issued with instance variable & local variable names

String

- A sequence of characters
- A String is a built-in Java object type
- Java provides this type because it’s used so frequently
- Examples of String creation:

  String s1 = "hello"; //shortcut
  OR String s1 = new String("hello");

  String s2 = s1 + "world";
  System.out.println(s2);
String (contd..)

```
String c1 = new String("Hill");
```

Stack and Heap

```
10
```

“Has a” Relationship

- An object of type A has an instance variable which is an object whose type is B. (A “has a” B.)
- E.g: A Freshman object whose room is of reference type DormRoom

```
DormRoom
```

```
Freshman
```

The UML diagrams below show instance variables and methods of Freshman and DormRoom object:

- UML (Universal Modeling Language) industry standard used to describe classes in OOP

DormRoom Code

```
public class DormRoom{
  private int num;
  private String bldgName;

  public DormRoom(int n, String b){
    num = n;
    bldgName = b;
  }

  public String getLocation(){
    return num + " " + bldgName;
  }
}
```

A DormRoom on the Heap

```
DormRoom room = new DormRoom(208, "Hill");
System.out.println(room.getLocation());
```

```
208 Hill
```

```
DormRoom
```

```
room
```

```
DormRoom
```

```
Stack
```

```
HEAP
```

```
DormRoom room = new DormRoom(208, "Hill");
System.out.println(room.getLocation());
```

```
208 Hill
```
Freshman Code

```java
DormRoom room = new DormRoom(208, "Hill");
Freshman f = new Freshman("jo", room);
System.out.println(f.getName());
System.out.println(f.getRoom().getLocation());
```

```java
public class Freshman{
    private String name;
    private DormRoom room;
    public Freshman(String n, DormRoom r){
        name = n;
        room = r;
    }
    public String getName(){ return name;}
    public DormRoom getRoom(){ return room;}
}
```

Note on Dot Operator and Parentheses

- Dot operator and parentheses for methods calls have same precedence
- Associativity is from L to R
- `f.getRoom().getLocation()` is equivalent to `(f.getRoom()).getLocation()`

A Freshman on the Heap

```
DormRoom room = new DormRoom(208, "Hill");
Freshman f = new Freshman("jo", room);
System.out.println(f.getName());
System.out.println(f.getRoom().getLocation());
```

More methods to Freshman

```
public class Freshman{
    ...
    public void changeRoom(DormRoom r){
        room = r;
    }
    public String address(){
        return room.getLocation();
    }
    public boolean hasARoom(){
        if(room != null)
            return true;
        else
            return false;
    }
}
```
More Interactions

```java
DormRoom r = new DormRoom(176, "McNair");
f.changeRoom(r);
System.out.println(f.getRoom().getLocation());
176 McNair
System.out.println(f.getAddress());
176 McNair
System.out.println(f.hasARoom());
true
DormRoom rr = null; // rr is null
f.changeRoom(rr);
System.out.println(f.hasARoom());
false
```

Garbage Collection

- Every time the new keyword is used
  - An object created and space for it allocated on the Heap section
- Sun Microsystems Java provides a way to clean up the space with unused objects
  - Known as Garbage Collection
  - However Javelin does not support it
    - So be conservative in allocating objects