

Introduction to Programming

with Java, for Beginners

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

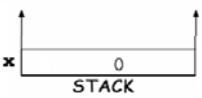
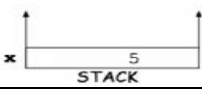
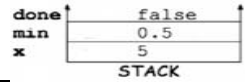
2

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
- In Dr Java's interaction pane storage information exists as long as RESET button is not clicked

3

How the Stack Works

DrJava Interaction (same applies for methods)	Stack
> int x = 0;	 <p>A diagram of a stack with two vertical arrows pointing outwards. Inside, a horizontal bar represents a memory slot. To the left of the bar is the label 'x'. Inside the bar is the value '0'. Below the bar is the label 'STACK'.</p>
> x = 5;	 <p>A diagram of a stack with two vertical arrows pointing outwards. Inside, a horizontal bar represents a memory slot. To the left of the bar is the label 'x'. Inside the bar is the value '5'. Below the bar is the label 'STACK'.</p>
double min = 0.5; boolean done = false;	 <p>A diagram of a stack with two vertical arrows pointing outwards. Inside, three horizontal bars represent memory slots. To the left of the top bar is the label 'done', and inside the bar is the value 'false'. To the left of the middle bar is the label 'min', and inside the bar is the value '0.5'. To the left of the bottom bar is the label 'x', and inside the bar is the value '5'. Below the bars is the label 'STACK'.</p>

Note: Variables are added in the order they are declared

4

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

5

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
 - p.getName()
 - mario.getStrength()

6

Value of a Reference Variable

Example:

```
> Counter c1;
> c1
null
> c1 = new Counter();
> c1
Counter@e05ad6
```

- e05ad6 is location in memory where object that c1 is pointing resides
 - e05ad6 hexadecimal (base 16) number
 - This location will differ on your computer
- We don't have to (and can't) deal with these hex numbers directly
 - Convenience of using variables

7

How the Heap Works

DrJava Interactions	Stack and Heap
<pre>> int x = 99; > Counter c1; > c1 null</pre>	
<pre>> c1 = new Counter(); > c1 Counter@2f996f</pre>	
<pre>> c1.incrementCount(); > Counter c2 = new Counter(); > c2 Counter@4a0ac5</pre>	

8

Aliases

- Two or more references can point to the same object
 - These references are then known as aliases
- Example (In Dr Java Interactions Pane)

```
> Student s1= new Student("Lisa", 5);
> s1
Student@83d8be
> Student s2 = s1;
> s2
Student@83d8be
> s1.getAge()
5
> s2.getAge()
5
```

9

Scope Issues with variables

```
public class Dot{
    private int x;
    private int y;

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

Local variable **x & y** shadows the instance variable **x & y**

Solution →

```
public class Dot{
    private int x;
    private int y;

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

10

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
 - Cannot be modified (i.e. it is final)
 - Used to Differentiate objects & avoid scope issued with instance variable & local variable names

11

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 = new String("hello");
> String s2 = "hello"; // commonly used shortcut
> s2 + " you!"
"hello you!"
> s2 = "The result is " + 100;
> System.out.println(s2);
"The result is 100"
```

12

String (contd..)

DrJava Interactions	Stack and Heap
<pre>> String c1 = new String("Hill");</pre>	

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

Note:

- We do not get heap address of String reference
- Later, it will be clear why this case..

13

Comparing Strings

- If the == operator is used
 - Java compares the addresses where the String objects are stored, not the letters in the String
 - For example:


```
> String a = "hi";
> String b = "hi";
> a == b
> false
```
- Use the String class' *equals* method to compare two Strings for equality

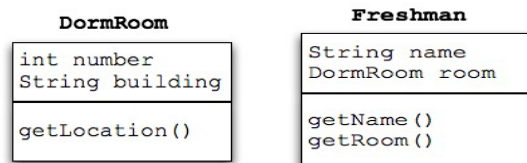

```
> a.equals(b)
true
> b.equalsIgnoreCase("HI")
true
```

Just like the Math class, String class is part of Java Language & hence directly used

14

"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



- The *UML diagrams* below show instance variables and methods of Freshman and DormRoom object:
 - UML(Universal Modeling Lanaguage) industry standard used to describe classes in OOP

15

DormRoom Code

```

DormRoom
int number
String building
getLocation()

> DormRoom room = new DormRoom(208, "Hill");
> room.getLocation()
"208 Hill"

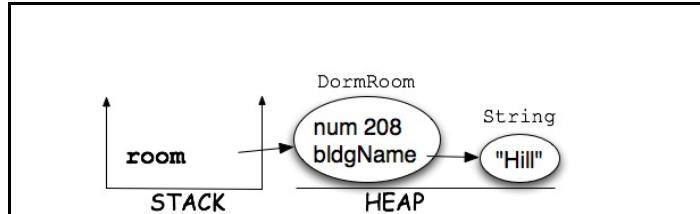
public class DormRoom{
    private int num;
    private String bldgName;

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

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

16

A DormRoom on the Heap



```
> DormRoom room = new DormRoom(208, "Hill");
> room.getLocation()
"208 Hill"
```

17

Freshman Code

```
Freshman
String name
DormRoom room
getName()
getRoom()
```

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

```
public class Freshman{
    private String name;
    private DormRoom room;

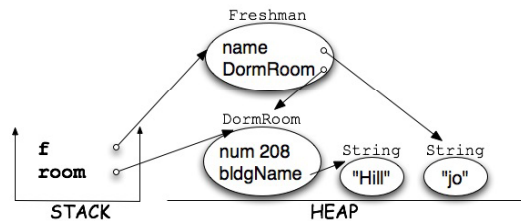
    public Freshman(String n, DormRoom r){
        name = n;
        room = r;
    }

    public String getName(){ return name;}
    public DormRoom getRoom(){ return room;}
}
```

18

A Freshman on the Heap

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



19

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;
    }
}
```

20

More Interactions

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

> DormRoom r = new DormRoom(176, "McNair");
> f.changeRoom(r);
> f.getRoom().getLocation()
"176 McNair"
> f.address()
"176 McNair"
> f.hasARoom()
true

> DormRoom rr; //rr is null
> f.changeRoom(rr);
> f.hasARoom()
false
> f.getRoom().getLocation()
// Error - Why?
```