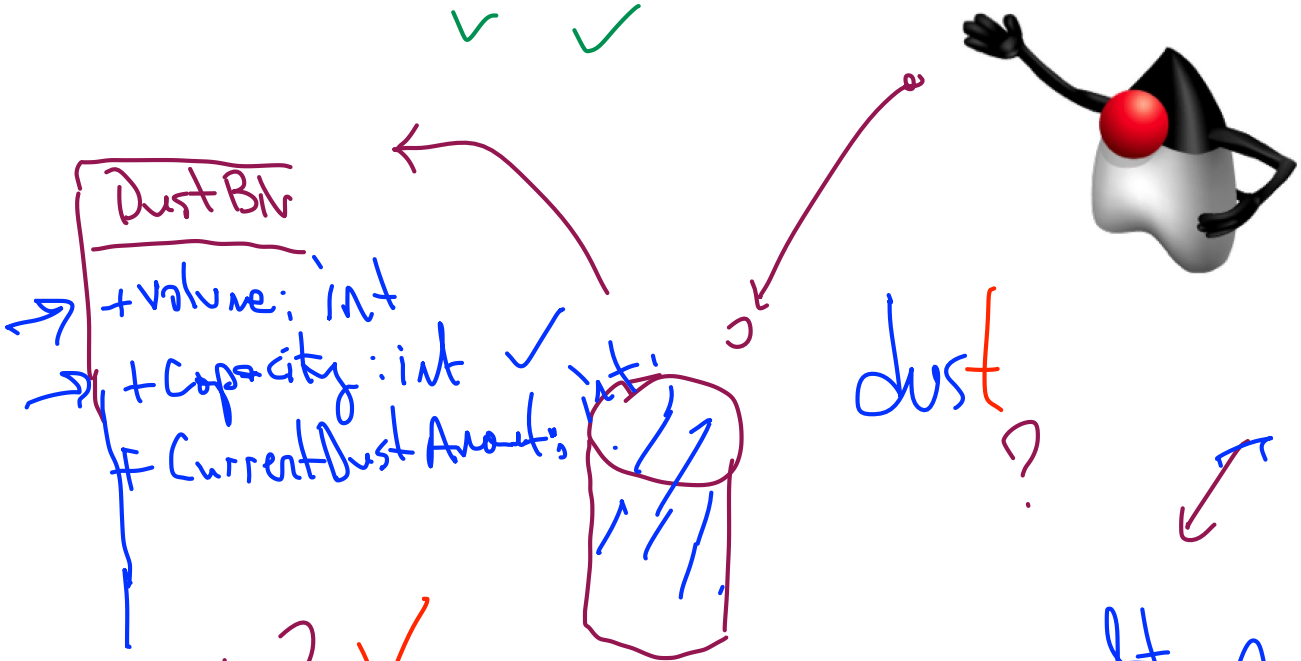
Java Language Review

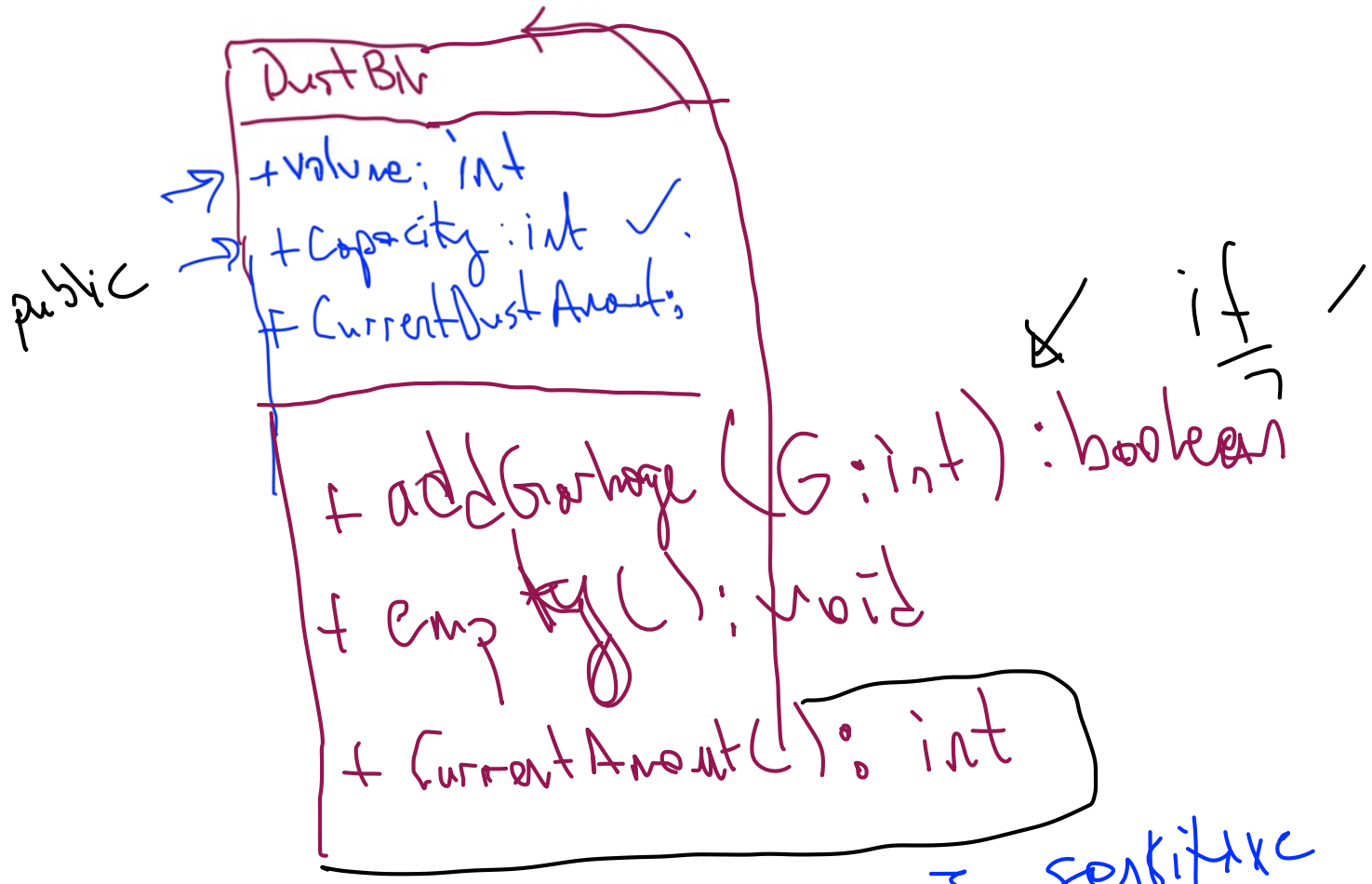
This lesson is a review of fundamental Java and programming concepts. It is assumed that students are familiar with the following concepts:

- The basic structure of a Java class
- Program block and comments
- Variables
- Basic `if-else` and `switch` branching constructs
- Iteration with `for` and `while` loops



Requirements?

- Every dustbin has volume, and
- it should tell how much full it is
- I should be able to add some rubbish method
- I should be able to empty dustbin method



Java is sensitive to case

optional

capital

```
[public] class DustBin {
    public int Capacity=10;
    public int CurrGarbageAmount=0;

    public int CurrentAmount() {
        return currGarbageAmount;
    }
}
```

small camel case

DustBin.java

attributes / fields

behaviours / methods

```
public void Empty(){
    CurrGarbageAmount = 0; } //empty
public boolean addGarbage(int Gar){
  if( CurrGarbageAmount + Gar > capacity)
  { System.out.println("Not enough space");
    return false
  } //if
  else
  {     CurrGarbageAmount = CurrGarbageAmount +
                            Gar;
  return true; }
} //end of DustBin class
```

Design the class

Driver class

Default Constructor

no argument

```
public DustBin(){ }
```

```
public class Driver {

public static void main(String[]
args ){

    DistBin myDB= new DistBM();

    }

}
```

constructor

Class Structure

```
package <package_name>;

import <other_packages>;

public class ClassName {
    <variables(also known as fields)>;

    <constructor method(s)>;

    <other methods>;
}
```

A Simple Class

A simple Java class with a main method:

```
public class Simple {

    public static void main(String args[]){

    }
}
```

Code Blocks

- Every class declaration is enclosed in a code block.
- Method declarations are enclosed in code blocks.
- Java fields and methods have block (or class) scope.
- Code blocks are defined in braces:

```
{ }
```

- Example:

```
public class SayHello {
    public static void main(String[] args) {
        System.out.println("Hello world");
    }
}
```

fields

methods [constructor(s)

other methods

Primitive Data Types

Integer	Floating Point	Character	True False
byte short int long	float double	char	boolean
1, 2, 3, 42 07 0xff	3.0F .3337F 4.022E23	'a' '\u0061' '\n'	true false
0	0.0	'\u0000'	false

Append uppercase or lowercase "L" or "F" to the number to specify a long or a float number.

Java SE 7 Numeric Literals

In Java SE 7 (and later versions), any number of underscore characters (_) can appear between digits in a numeric field. This can improve the readability of your code.

```
long creditCardNumber = 1234_5678_9012_3456L;
long socialSecurityNumber = 999_99_9999L;
long hexBytes = 0xFF_EC_DE_5E;
long hexWords = 0xCAFE_BABE;
long maxLong = 0x7fff_ffff_ffff_ffffL;
byte nybbles = 0b0010_0101;
long bytes = 0b11010010_01101001_10010100_10010010;
```

Java SE 7 Binary Literals

In Java SE 7 (and later versions), binary literals can also be expressed using the binary system by adding the prefixes `0b` or `0B` to the number:

```
// An 8-bit 'byte' value:
byte aByte = 0b0010_0001;

// A 16-bit 'short' value:
short aShort = (short)0b1010_0001_0100_0101;

// Some 32-bit 'int' values:
int anInt1 = 0b1010_0001_0100_0101_1010_0001_0100_0101;
int anInt2 = 0b101;
int anInt3 = 0B101; // The B can be upper or lower case.
```

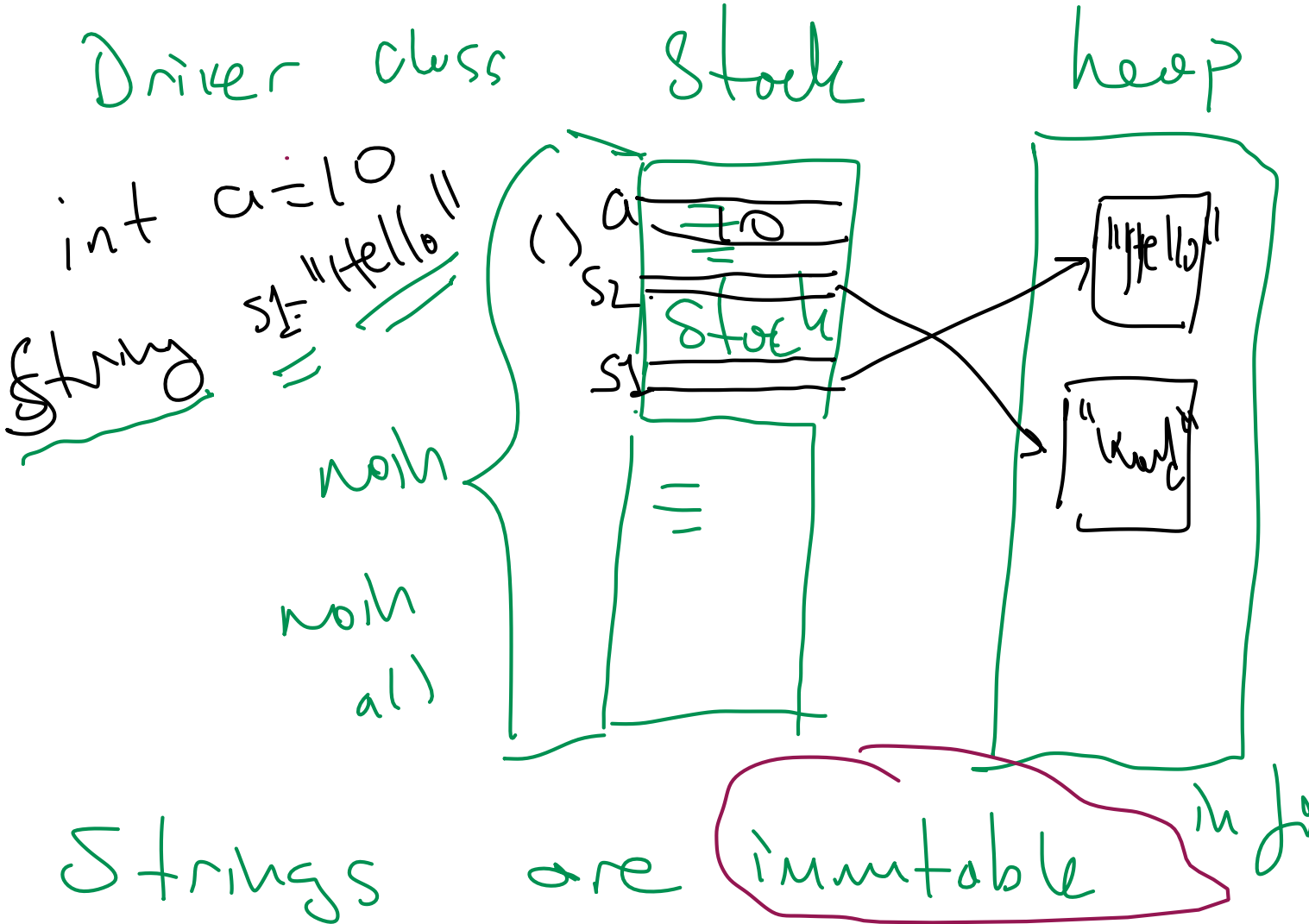
Operators

- Simple assignment operator
  - = Simple assignment operator
- Arithmetic operators
  - \+ Additive operator (also used for String concatenation)
  - – Subtraction operator
  - * Multiplication operator
  - / Division operator
  - % Remainder operator
- Unary operators
  - \+ Unary plus operator; indicates positive
  - – Unary minus operator; negates an expression
  - ++ Increment operator; increments a value by 1
  - -- Decrement operator; decrements a value by 1
  - ! Logical complement operator; inverts the value of a boolean

Strings

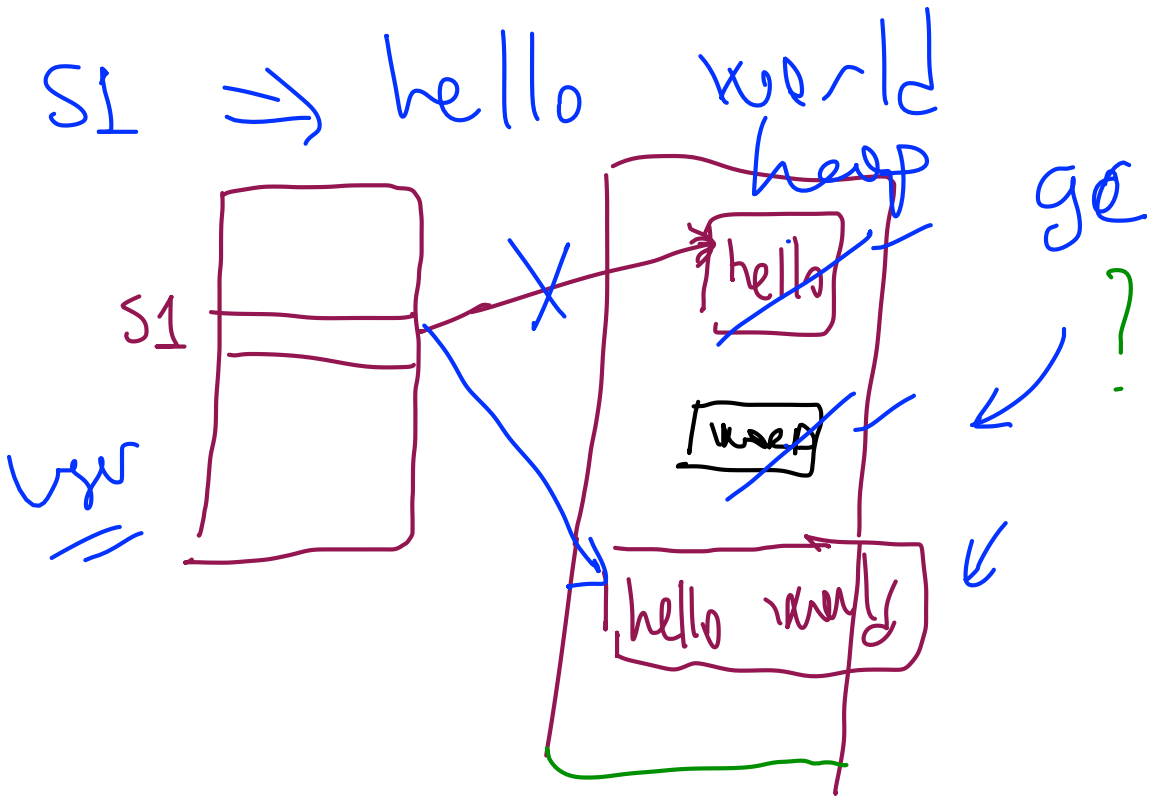
```
1 public class Strings {
2
3     public static void main(String args[]){
4
5         char letter = 'a';
6
7         String string1 = "Hello";
8         String string2 = "World";
9         String string3 = "";
10        String dontDoThis = new String ("Bad Practice");
11
12        string3 = string1 + string2; // Concatenate strings
13
14        System.out.println("Output: " + string3 + " " + letter);
15
16    }
17 }
```

String literals are also String objects.



immutable : can not be change

S1 = S1 + "WORLD" ;

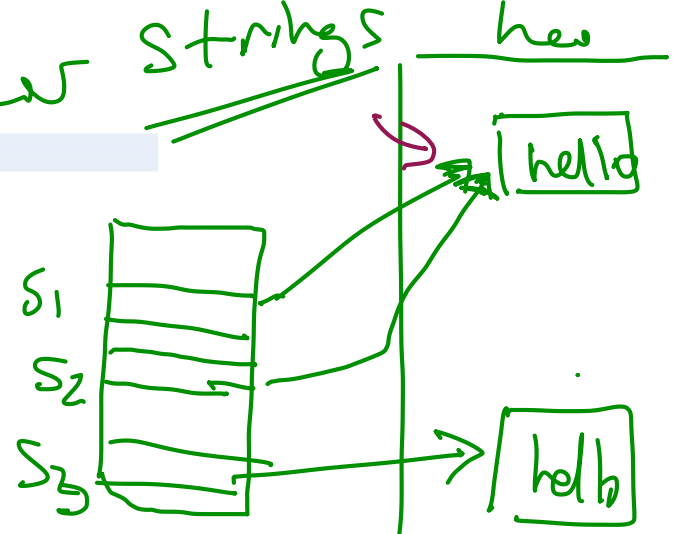


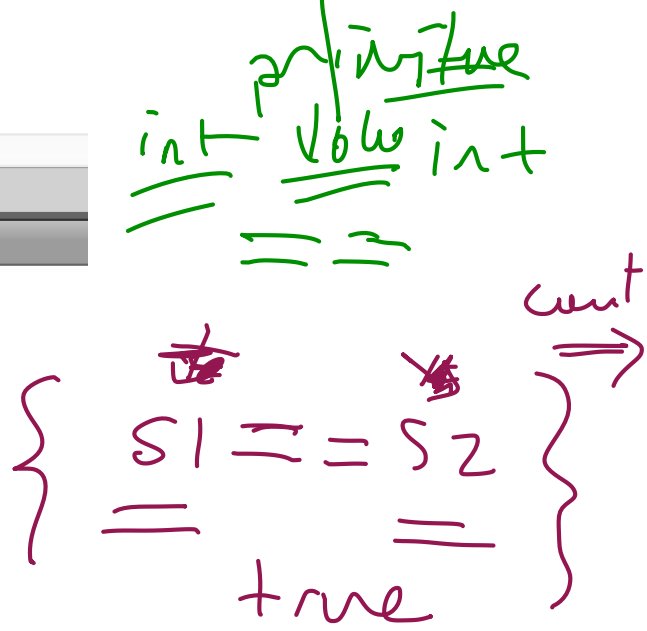
```java
public static void main(String[] args) {
    // TODO code application logic here
    System.out.println("Hello ");

    String s1= "Hello";
    String s2= "Hello";
    String s3= new String("Hello");


    System.out.println(s1==s2);
    System.out.println(s1==s3);

/*
```

week2projects.Week2Projects › main

tput – Week2Projects (run) ×

```
run:
Hello
true
false
BUILD SUCCESSFUL (total time: 2 seconds)
```





String Operations

```
1 public class StringOperations {
2     public static void main(String arg[]){
3         String string2 = "World";
4         String string3 = "";
5
6         string3 = "Hello".concat(string2);
7         System.out.println("string3: " + string3);
8
9         // Get length
10        System.out.println("Length: " + string1.length());
11
12        // Get SubString
13        System.out.println("Sub: " + string3.substring(0, 5));
14
15        // Uppercase
16        System.out.println("Upper: " + string3.toUpperCase());
17    }
18}
```

String literals are automatically created as String objects if necessary.

if else

```
1 public class IfElse {
2
3     public static void main(String args[]){
4         long a = 1;
5         long b = 2;
6
7         if (a == b){
8             System.out.println("True");
9         } else {
10            System.out.println("False");
11        }
12
13    }
14 }
```

Logical Operators

- Equality and relational operators
  - == Equal to
  - != Not equal to
  - > Greater than
  - >= Greater than or equal to
  - < Less than
  - <= Less than or equal to
- Conditional operators
  - && Conditional-AND
  - || Conditional-OR
  - ?: Ternary (shorthand for `if-then-else` statement)
- Type comparison operator
  - `instanceof` Compares an object to a specified type

a = 5; b = 20;

a = a > b ? 5 : 10;

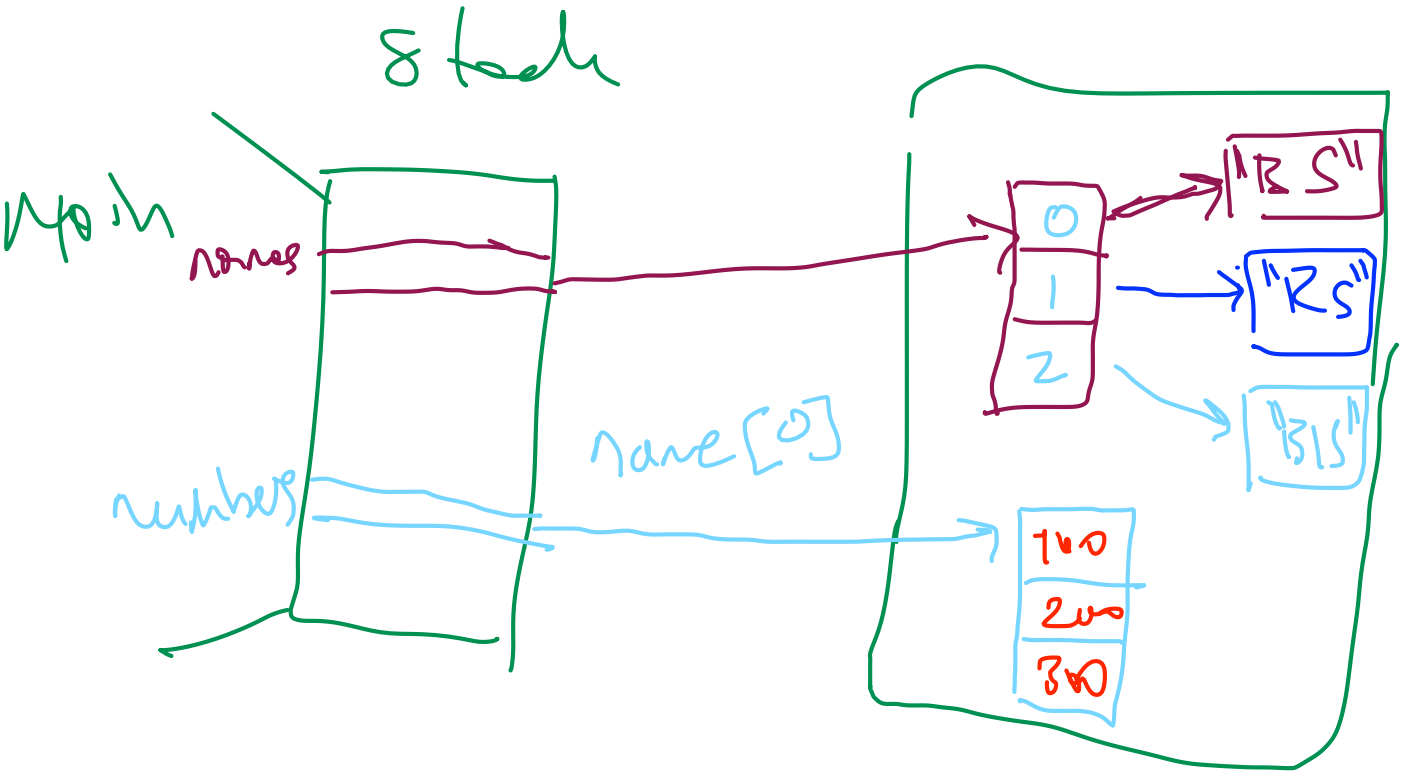
T F

a = 10;

Arrays and for-each Loop

```
1 public class ArrayOperations {
2     public static void main(String args[]){
3
4         String[] names = new String[3];
5
6         names[0] = "Blue Shirt";
7         names[1] = "Red Shirt";
8         names[2] = "Black Shirt";
9
10          int[] numbers = {100, 200, 300};
11
12        for (String name:names){
13            System.out.println("Name: " + name);
14        }
15
16          for (int number:numbers){
17              System.out.println("Number: " + number);
18          }
19    }
20 }
```

Arrays are objects. Array objects have a final field length.



for Loop

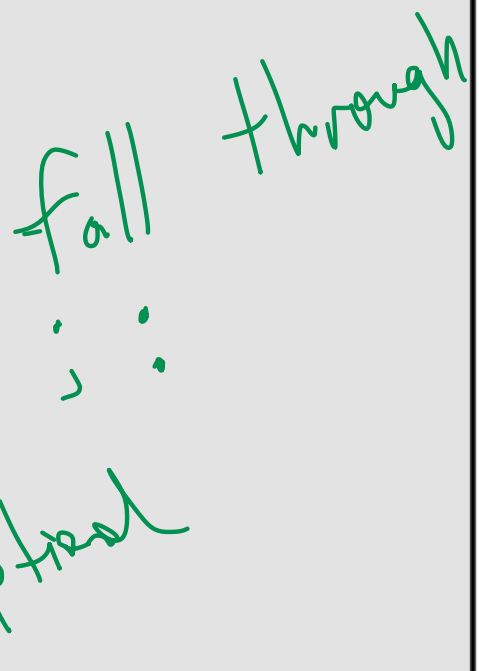
```
1 public class ForLoop {
2
3     public static void main(String args[]){
4
5         for (int i = 0; i < 9; i++){
6             System.out.println("i: " + i);
7         }
8
9     }
10 }
```

while Loop

```
1 public class WhileLoop {
2
3    public static void main(String args[]){
4
5       int i = 0;
6       int[] numbers = {100, 200, 300};
7
8       while (i < numbers.length ){
9          System.out.println("Number: " + numbers[i]);
10          i++;
11       }
12    }
13 }
```

String `switch` Statement

```
1 public class SwitchStringStatement {
2    public static void main(String args[]){
3
4        String color = "Blue";
5        String shirt = " Shirt";
6
7        switch (color){
8            case "Blue":
9                shirt = "Blue" + shirt;
10               break;
11           case "Red":
12               shirt = "Red" + shirt;
13               break;
14           default:
15               shirt = "White" + shirt;
16       }
17
18       System.out.println("Shirt type: " + shirt);
19   }
20 }
```



Java Naming Conventions

```
1 public class CreditCard {
2     public final int VISA = 5001;
3     public String accountName;
4     public String cardNumber;
5     public Date expDate;
6
7     public double getCharges(){
8         // ...
9     }
10
11    public void disputeCharge(String chargeId, float amount){
12        // ...
13    }
14}
```

Class names are nouns in upper camel case.

Constants should be declared in all uppercase. letters

Variable names are short but meaningful in lower camel case.

Methods should be verbs, in lower camel case.

final

variable

Constant

{ method ⇒ can't be overridden

Class ⇒ can't have subclasses

A Simple Java Class: `Employee`

A Java class is often used to represent a concept.

```
1 package com.example.domain;
2 public class Employee {
3     public int empId;
4     public String name;
5     public String ssn;
6     public double salary;
7
8     public Employee () {
9     }
10
11     public int getEmpId () {
12         return empId;
13     }
14 }
```

class declaration

fields

a constructor

a method

Methods

When a class has data fields, a common practice is to provide methods for storing data (setter methods) and retrieving data (getter methods) from the fields.

```
1 package com.example.domain;
2 public class Employee {
3     public int empId;
4     // other fields...
5     public void setEmpId(int empId) {
6         this.empId = empId;
7     }
8     public int getEmpId() {
9         return empId;
10    }
11    // getter/setter methods for other fields...
12 }
```

Often a pair of methods to set and get the current field value.

"this" is new for you

this keyword ?

object its self "this"

Creating an Instance of an Object

To construct or create an instance (object) of the `Employee` class, use the `new` keyword.

```
/* In some other class, or a main method */
Employee emp = new Employee();
emp.empId = 101;    // legal if the field is public,
                    // but not good OO practice
emp.setEmpId(101); // use a method instead
emp.setName("John Smith");
emp.setSsn("011-22-3467");
emp.setSalary(120345.27);
```

Invoking an instance method.

- In this fragment of Java code, you construct an instance of the `Employee` class and assign the reference to the new object to a variable called `emp`.
- Then you assign values to the `Employee` object.

Constructors

```
public class Employee {
    public Employee() {
    }
}
```

A simple no-argument (no-arg) constructor.

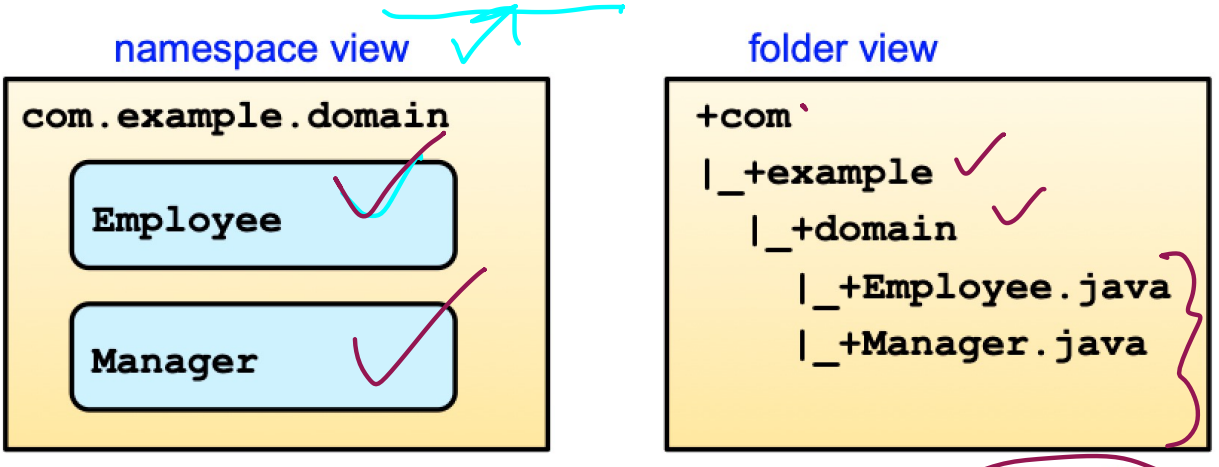
```
Employee emp = new Employee();
```

- A constructor is used to create an instance of a class.
- Constructors can take parameters.
- A constructor that takes no arguments is called a *no-arg* constructor.

default constructor.

package Statement

The `package` keyword is used in Java to group classes together. A package is implemented as a folder and, like a folder, provides a *namespace* to a class.



Always declare a package!

`import` Statements

The `import` keyword is used to identify classes you want to reference in your class.

- The `import` statement provides a convenient way to identify classes that you want to reference in your class.

```
import java.util.Date;
```

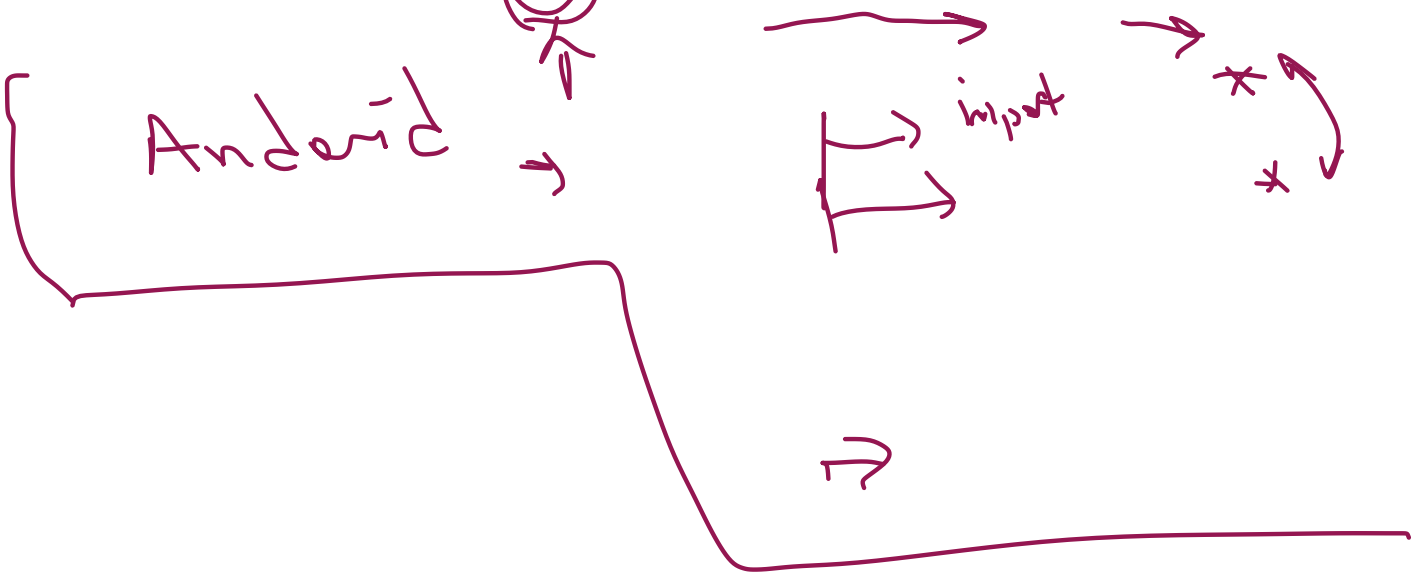
- You can import a single class or an entire package:

```
import java.util.*;
```

- You can include multiple `import` statements:

```
import java.util.Date;
import java.util.Calendar;
```

- It is good practice to use the full package and class name rather than the wildcard `*` to avoid class name conflicts.



More on `import`

- Import statements follow the package declaration and precede the class declaration.
- An import statement is not required.
- By default, your class always imports `java.lang.*`
- You do not need to import classes that are in the same package:

```
package com.example.domain;
import com.example.domain.Manager; // unused import
```

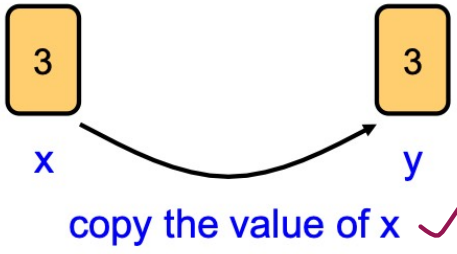
Java Is Pass-By-Value

The Java language (unlike C++) uses pass-by-value for all assignment operations.

- To visualize this with primitives, consider the following:

```
int x = 3;
int y = x;
```

- The value of `x` is copied and passed to `y`:



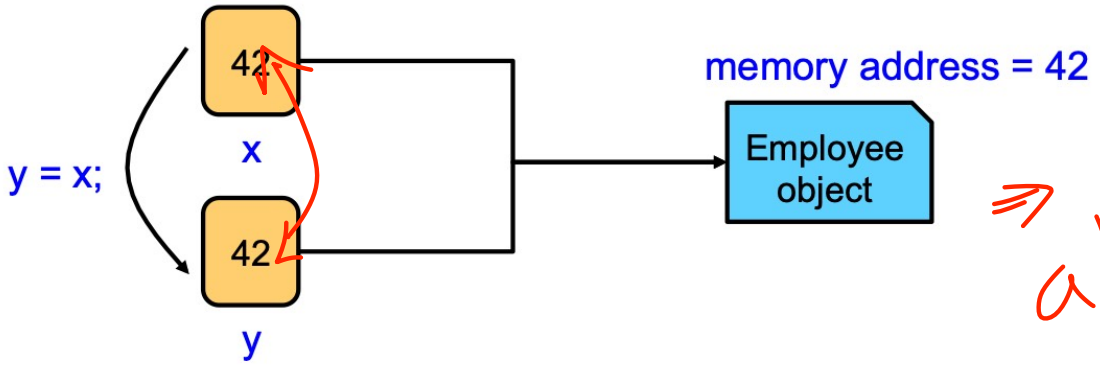
- If `x` is later modified (for example, `x = 5;`), the value of `y` remains unchanged.

Pass-By-Value for Object References

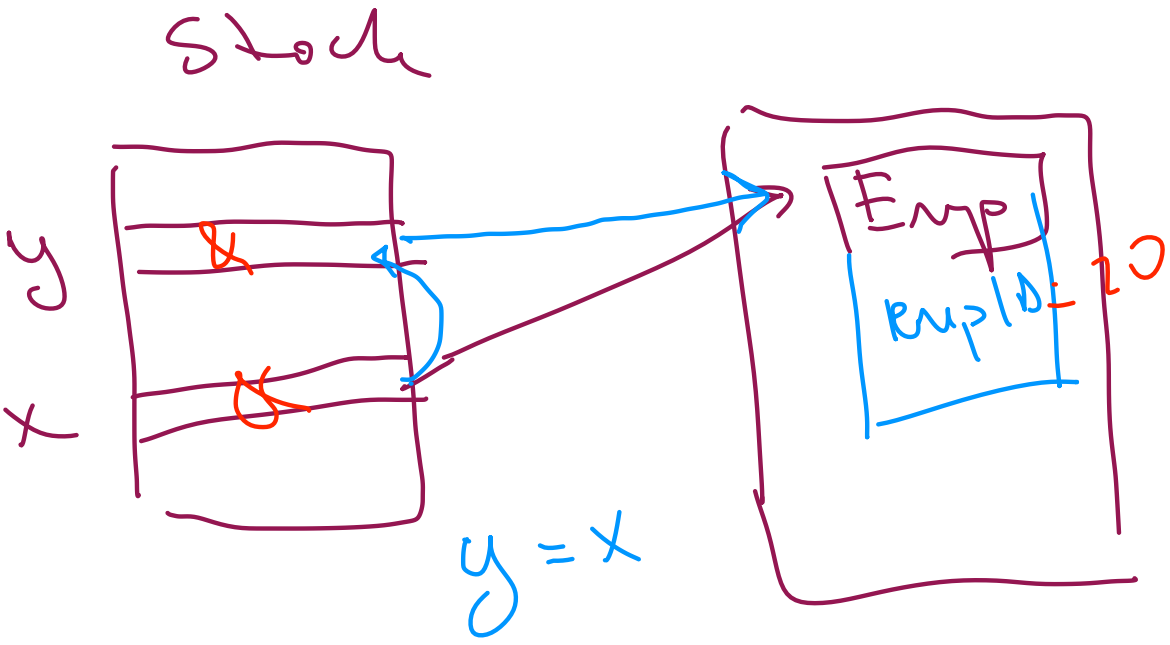
For Java objects, the *value* of the right side of an assignment is a reference to memory that stores a Java object.

```
Employee x = new Employee();
Employee y = x;
```

- The reference is some address in memory.



- After the assignment, the value of `y` is the same as the value of `x`: a reference to the same `Employee` object.



y.empID =

x.empID = 20

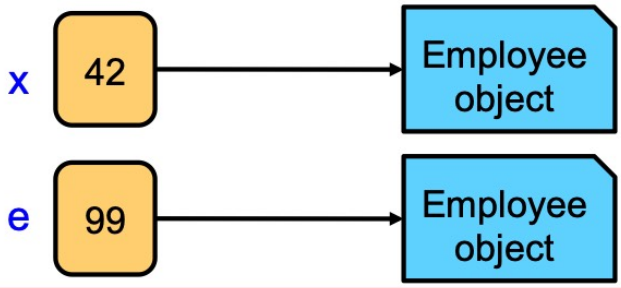
Objects Passed as Parameters

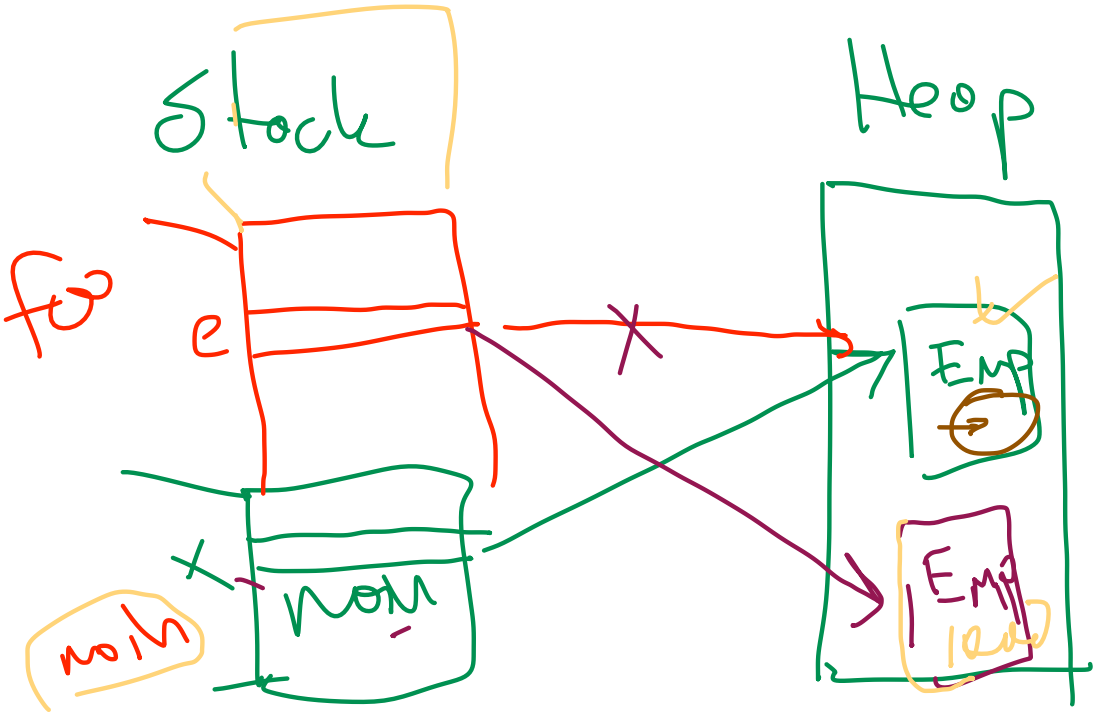
- Whenever a new object is created, a new reference is created. Consider the following code fragments:

```
Employee x = new Employee();
foo(x);
```

```
public void foo(Employee e) {
    e = new Employee();
    e.setSalary (1_000_000.00);  // What happens to x here?
}
```

- The value of `x` is unchanged as a result of the method call `foo`:





How to Compile and Run

Java class files must be compiled before running them.
To compile a Java source file, use the Java compiler (`javac`).

```
javac -cp <path to other classes> -d <complier output
path> <path to source>.java
```

- You can use the `CLASSPATH` environment variable to the directory above the location of the package hierarchy.
- After compiling the source `.java` file, a `.class` file is generated.
- To run the Java application, run it using the Java interpreter (`java`):

```
java -cp <path to other classes> <package
name>.<classname>
```

Compiling and Running: Example

- Assume that the class shown in the notes is in the directory `D:\test\com\example`:

```
javac -d D:\test D:\test\com\example\HelloWorld.java
```

- To run the application, you use the interpreter and the fully qualified class name:

```
java -cp D:\test com.example.HelloWorld
Hello World!

java -cp D:\test com.example.HelloWorld Tom
Hello Tom!
```

- The advantage of an IDE like NetBeans is that management of the class path, compilation, and running the Java application are handled through the tool.

Java Class Loader

During execution of a Java program, the Java Virtual Machine loads the compiled Java class files using a Java class of its own called the "class loader" (`java.lang.ClassLoader`).

- The class loader is called when a class member is used for the first time:

```
public class Test {
    public void someOperation() {
        Employee e = new Employee();
        //...
    }
}
```

The class loader is called to "load" this class into memory.

```
Test.class.getClassLoader().loadClass("Employee");
```

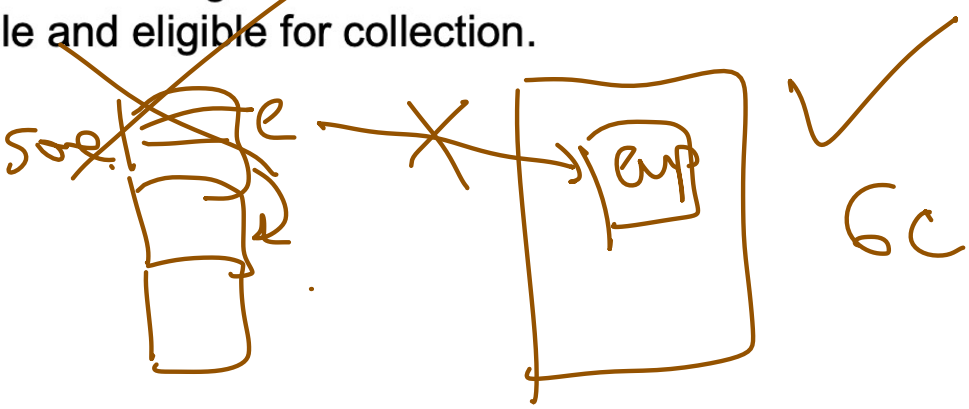
Garbage Collection

When an object is instantiated using the `new` keyword, memory is allocated for the object. The scope of an object reference depends on where the object is instantiated:

```
public void someMethod() {
    Employee e = new Employee();
    // operations on e
}
```

Object e scope ends here.

- When `someMethod` completes, the memory referenced by `e` is no longer accessible.
- Java's garbage collector recognizes when an instance is no longer accessible and eligible for collection.



Homework

1) Practice 2.1

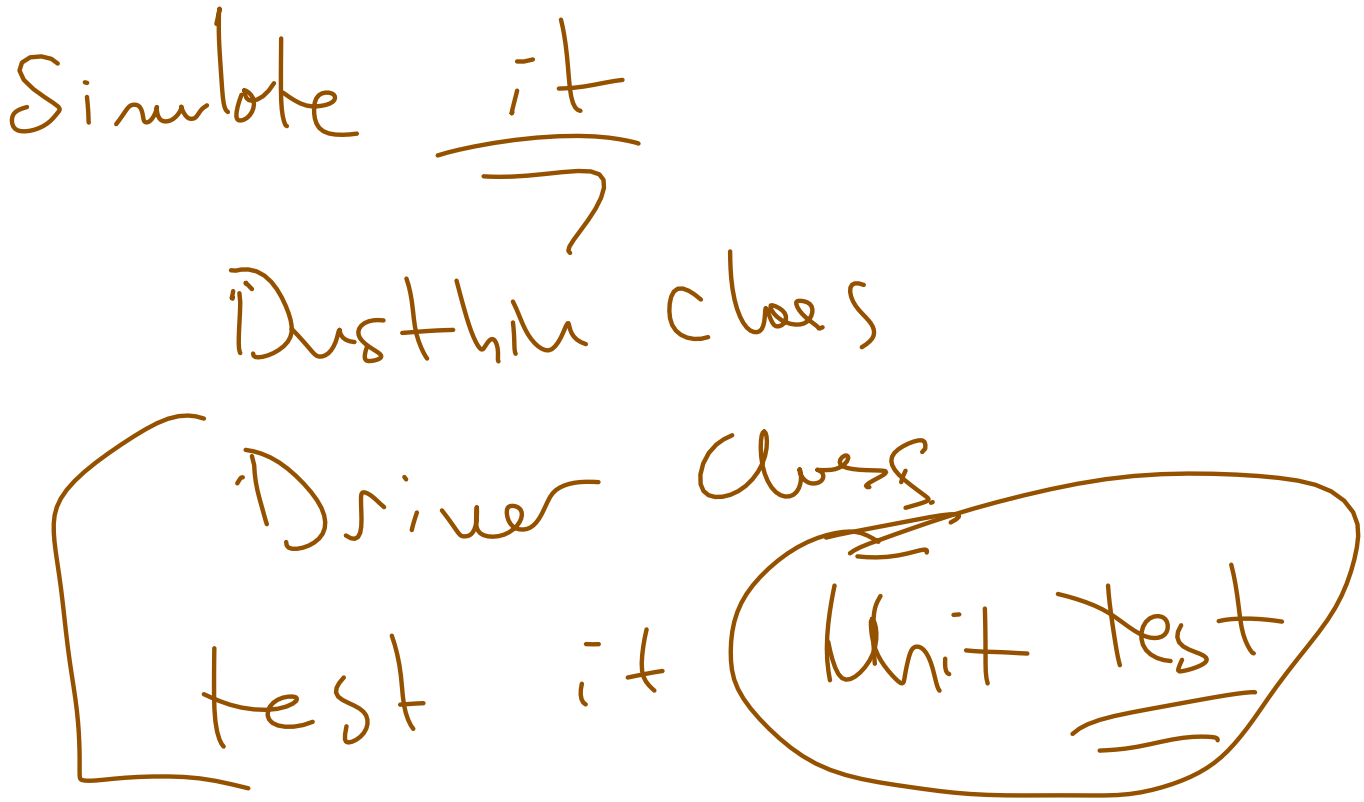
2) Use the Dustbin Class

+ But a a Lid

+ if you want to add garbage to dustbin you need to open it

+ Dustbin tells if lid is open or not

+ You can not empty a dustbin with a closed lid →


Simulate it
Dustin class
Driver class
test it
Unit test