Module 5
Reference Variables

Computer Science 2

Data representation

Primitive variables

- int
- char
- boolean
- double
- etc

Reference Variables

- String
- Arrays
- Objects (Scanner, Person, Animal, etc)

Initial values for primitive variables

```
byte var1;   // initial value = 0
short var2;  // initial value = 0
int var3;    // initial value = 0
long var4;   // initial value = 0L
float var5; // initial value = 0.0f
double var6; // initial value = 0.0d
char var7;   // initial value = '\u0000’
boolean flg; // initial value = false
Object var8; // initial value = null
String var9; // initial value = null
int[] var10; // initial value = null
```

Reference Variables

All **reference variables** (Objects, arrays, Strings) start with an initial value of null **`null`**.

What is null?

`null` is a special constant you can point to whenever you wish to point to the **absence of a value**.

Reference Variables

Any **reference variable** can be assigned a value of null. If we try to call an instance method of an object that has not been initialized, we will see the **`NullPointerException`** error.

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

    Scanner s1 = null;
   int x = s1.nextInt();

}
```

```
run:
Exception in thread "main" java.lang.NullPointerException
        at primitive_reference.Primitive_Reference.main(Primitive_Reference.java:21)
Java Result: 1
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
byte var1;
short var2;
int var3;
String var9;
int[] var10;
Object var11;
```

Variable name	Initial value	Memory location
byte var1	0	10F50
short var2	0	10F51
int var3	0	10F52
String var9	null	10F53
int[] var10	null	10F54
Object var11	null	10F55

```
var1 = 10;
var2 = 3;
var3 = 17;
var9 = "Hola";
```

Variable name	Initial value	Memory location
byte var1	10	10F50
short var2	3	10F51
int var3	17	10F52
String var9	11A30	10F53
	…	
	…	
var9[0]	H	11A30
var9[1]	o	11A31
var9[2]	l	11A32
var9[3]	a	11A33
var9[4]	end of String	11A34

```
var10 = new int[3];
var11 = new Object();
```

Variable name	Initial value	Memory location
int[] var10	12A00	10F54
Object var11	14FF1	10F55
	…	
var10[0]	0	12A00
var10[1]	0	12A01
var10[2]	0	12A02
	…	
Object.a	0	14FF1
Object.b	FALSE	14FF2
Object.c	0	14FF3

The difference between a primitive variable and a reference variable is now information is stored in memory.

Primitive variable contents store **information directly** (They have an already defined length!)

Reference variables **store a pointer to a memory location** where the data is stored. This is because the size of the data is defined during runtime.

Variable name	Initial value	Memory location
byte var1	10	10F50
short var2	3	10F51
int var3	17	10F52
String var9	11A30	10F53
int[] var10	12A00	10F54
Object var11	14FF1	10F55
	…	
	…	
	…	
	…	
var9[0]	H	11A30
var9[1]	o	11A31
var9[2]	l	11A32
var9[3]	a	11A33
var9[4]	end of String	11A34
	…	
var10[0]	0	12A00
var10[1]	0	12A01
var10[2]	0	12A02
	…	
Object.a	0	14FF1
Object.b	FALSE	14FF2
Object.c	0	14FF3

“I call it my billion-dollar mistake(…). Null references were created in 1964 - how much have they cost? (...)

This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.”

Sir Charles Anthony Richard Hoare

Reference Variables

When comparing reference variables, the == operator does not work.

`null` is a reserved word.

```
int[] c1 = null;
int[] c2 = new int[0];

if (c1 == c2) {
    System.out.println("Iguales");
} else {
    System.out.println("Diferentes");
}
```

```
run:
Diferentes
```

Reference Variables

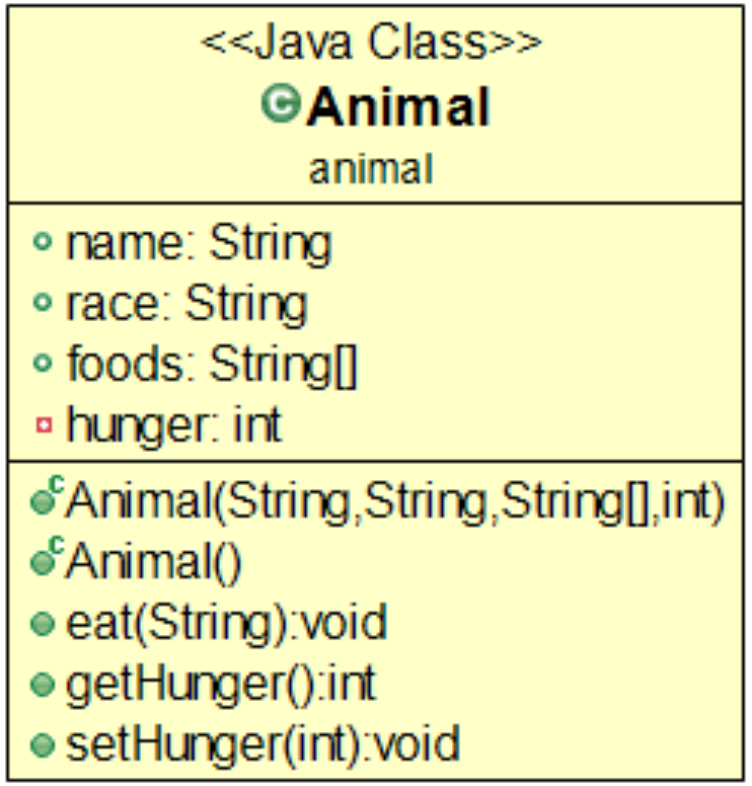
- You can void issues by checking pointer validity before using object attributes or methods.

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

    int[] x = null;
    printArray(x);
}

public static void printArray(int[] array) {

    //check pointer validity
    if (array == null){
        System.out.println("Imposible imprimir.");
        return;
    }

    for(int i = 0; i<array.length; i++) {
        System.out.println(array[i]);
    }
}
```

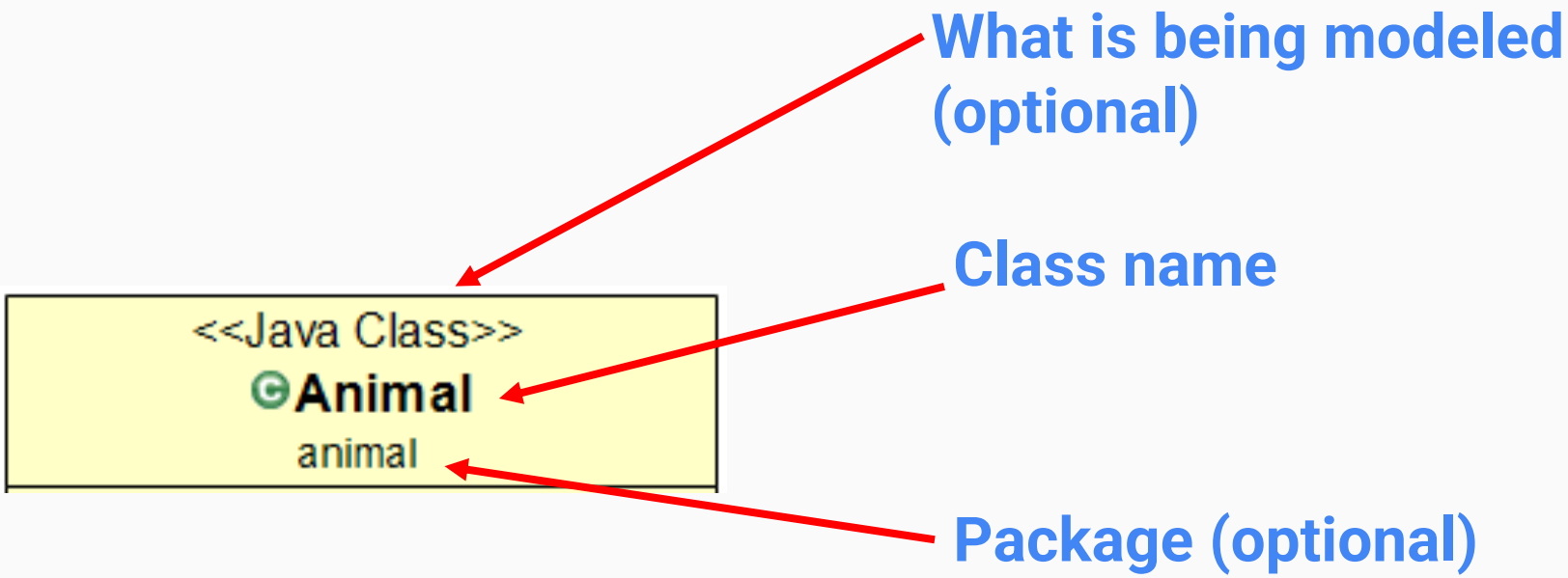
UML Notation

UML Notation

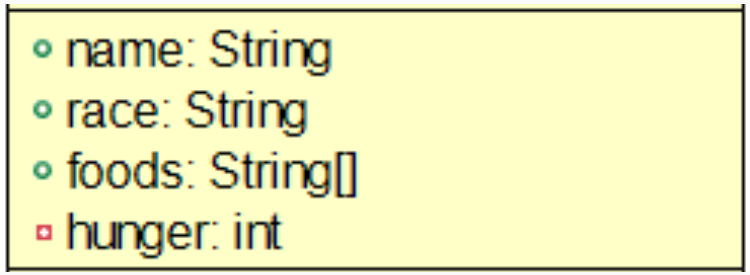
Unified Model Language is a series of visual standards to provide a way to represent the design of a system.




```java
public class Animal2 {

  public String name;
  public String race;
  public String[] foods;
  private int hunger;

  public Animal(String name, String race,
                String[] foods, int hunger) {}

  public Animal() {}

  public void eat(String inputFood) {}

  public int getHunger() {}

  public void setHunger(int hunger) {}

}
```

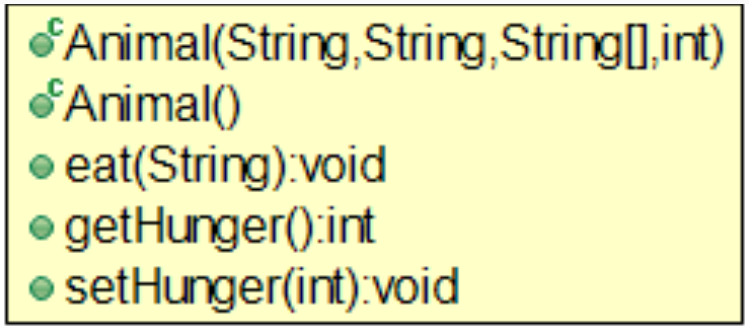

What is being modeled (optional)
Class name
<<Java Class>>
Animal
animal
Package (optional)

Attribute list



1. Access modifier
2. Variable name
3. :
4. Data type

Another way of representing the access modifiers is by using the plus and minus signs.
+ for public
- for private



Methods

1. Access modifier
2. Method name
3. Parameter list
4. :
5. Return value

The parameter list can also be specified using the following format

`name: data type`

Exercise

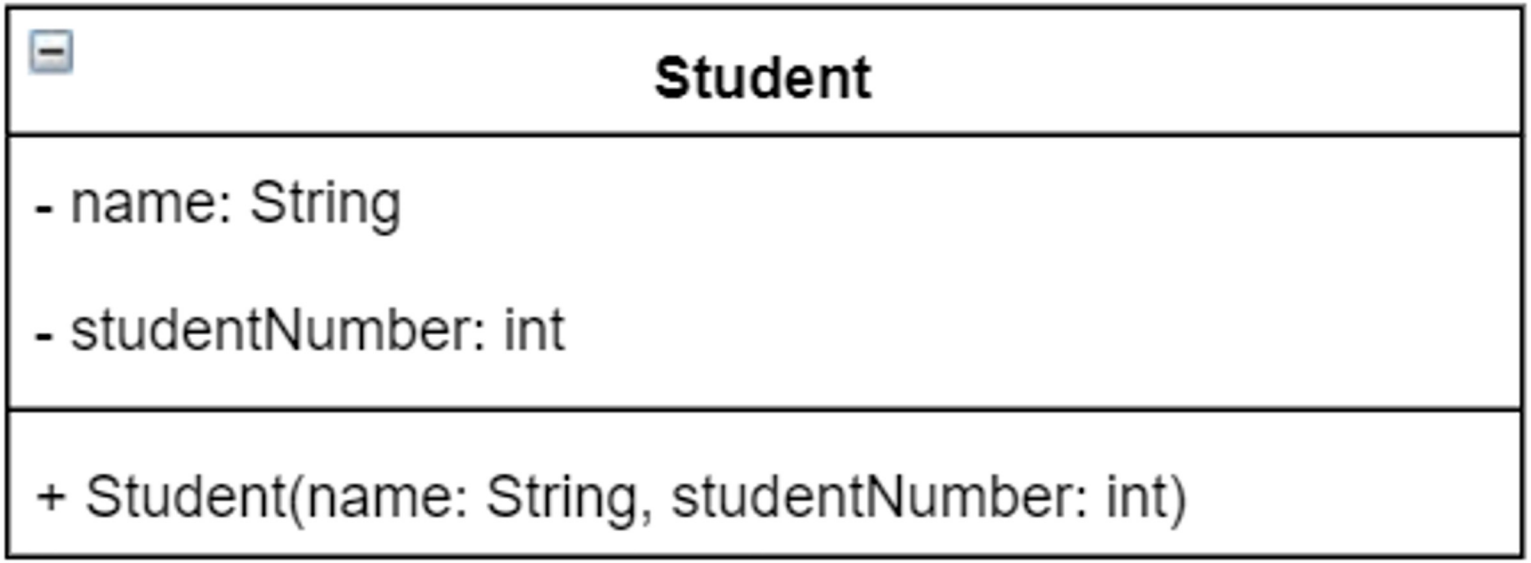
Student classroom

We want to represent a Classroom full of students.

Each student must be able to store a name and their student id.

Each Classroom must be able to hold an array of students, with the name of the class and the room where the class takes place.

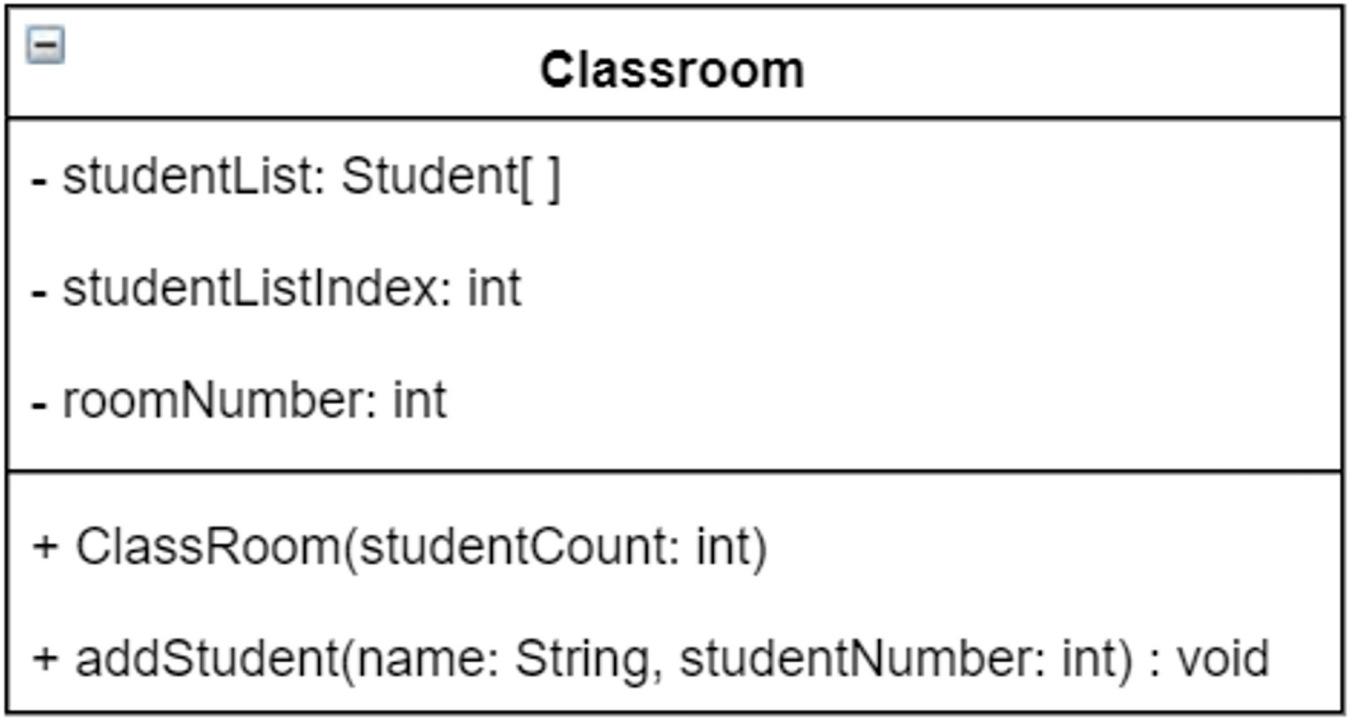
Exercise: Student classroom



Exercise: Student classroom

```
package student;

public class Student {

    private String name;
    private int studentNumber;

    public Student(String name, int studentNumber){
        this.name = name;
        this.studentNumber = studentNumber;
    }

}
```

Exercise: Student classroom



Exercise: Student classroom

```
package student;

public class Classroom {

    private Student[] studentList;
    private int studentListIndex;
    private int roomNumber;

    public Classroom(int studentCount, int roomNumber) {
        if (studentCount <= 0 || roomNumber < 0) {
            System.out.println("Error, invalid student count.");
            System.exit(0);
        }

        this.studentList = new Student[studentCount];
        this.studentListIndex = 0;
        this.roomNumber = roomNumber;
    }

    public void addStudent(String name, int studentNumber) {
        //only insert students into the list if it has space
        if (this.studentListIndex < this.studentList.length) {
            this.studentList[this.studentListIndex++] = new Student(name, studentNumber);
            System.out.println(name + " has been enrolled on the course!");
        } else {
            System.out.println("No space for " + name + "!");
        }
    }

}
```

Exercise: Student classroom

```
package student;

public class Test {

    public static void main(String[] args) {

        Classroom computerScience2 = new Classroom(5, 4303); //studentCount = 5, roomId = 4303
        computerScience2.addStudent("Omar", 1);
        computerScience2.addStudent("Jose", 2);
        computerScience2.addStudent("Martha", 3);
        computerScience2.addStudent("Eduardo", 4);
        computerScience2.addStudent("Karen", 5);
        computerScience2.addStudent("Mirthala", 6);

    }

}
```

OUTPUT

```
Omar has been enrolled on the course!
Jose has been enrolled on the course!
Martha has been enrolled on the course!
Eduardo has been enrolled on the course!
Karen has been enrolled on the course!
No space for Mirthala!
```