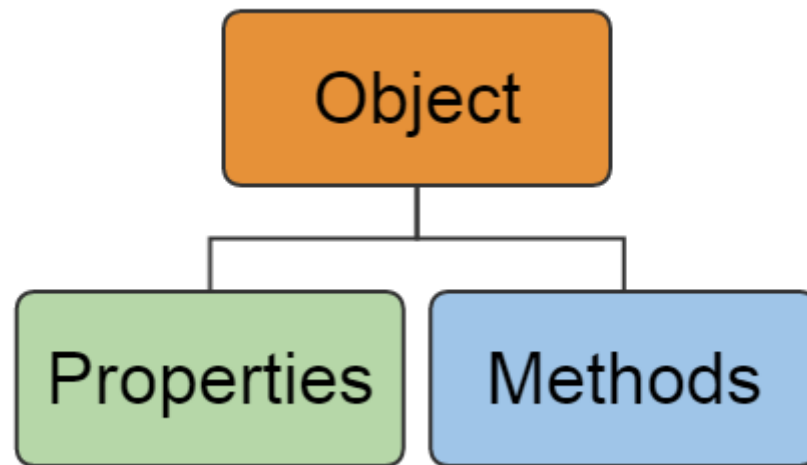




Object Oriented Programming

by Vlad Costel Ungureanu
for Learn Stuff IO



Object Oriented Programming Terminology

- ✓ **Program:** A multitude of object that interact with one another.
- ✓ **Package:** A folder name structure necessary for organizing code.
- ✓ **Class:** Prototype/blueprint which describes objects.
- ✓ **Object:** Entity described by state and behavior. An object is an instance of a Class.
- ✓ **Reference:** Entity which provides information for uniquely localizing an object in memory.
- ✓ **Interface:** A contract which imposes a specific behavior to a class.

Prototype of a Class

```
access_modifier [abstract][final] class ClassName [extends ParentClass] [implements Interface1 [, .. ]] {  
    // Variables  
    // Constructors  
    // Methods  
    // Imbricated classes (should be avoided, if possible)  
}
```

- ✓ All terms in brackets are optional
- ✓ The key word “extends” is used for inheritance
- ✓ A Java Class can implement any number of interfaces

Access modifiers

- ✓ **Private:** only accessible within the class
 - ✓ **Protected:** only accessible for class and sub classes
 - ✓ **Public:** accessible within the class and for any other class
 - ✓ **Implicit:** accessible at package level (for all classes in the same package)
-
- ✓ The implicit access modifier does not have to be specified. If no access modifier is explicitly declared the implicit one is assumed.

Static Keyword

- ✓ **Static variables:** values stored at class level not at instance level; are the same in all class instances.

Example: `static final double PI = 3.14;`

- ✓ **Static methods:** methods applicable at class level; not applicable at instance level; they can work only with parameters or other static class variables

Example: `double x = Math.sqrt(2);`

- ✓ **Static blocks:** constructor at class level not instance level; all static blocks are called before any class is instantiated

Example: `static {
 x = 2;
}`

Final Keyword

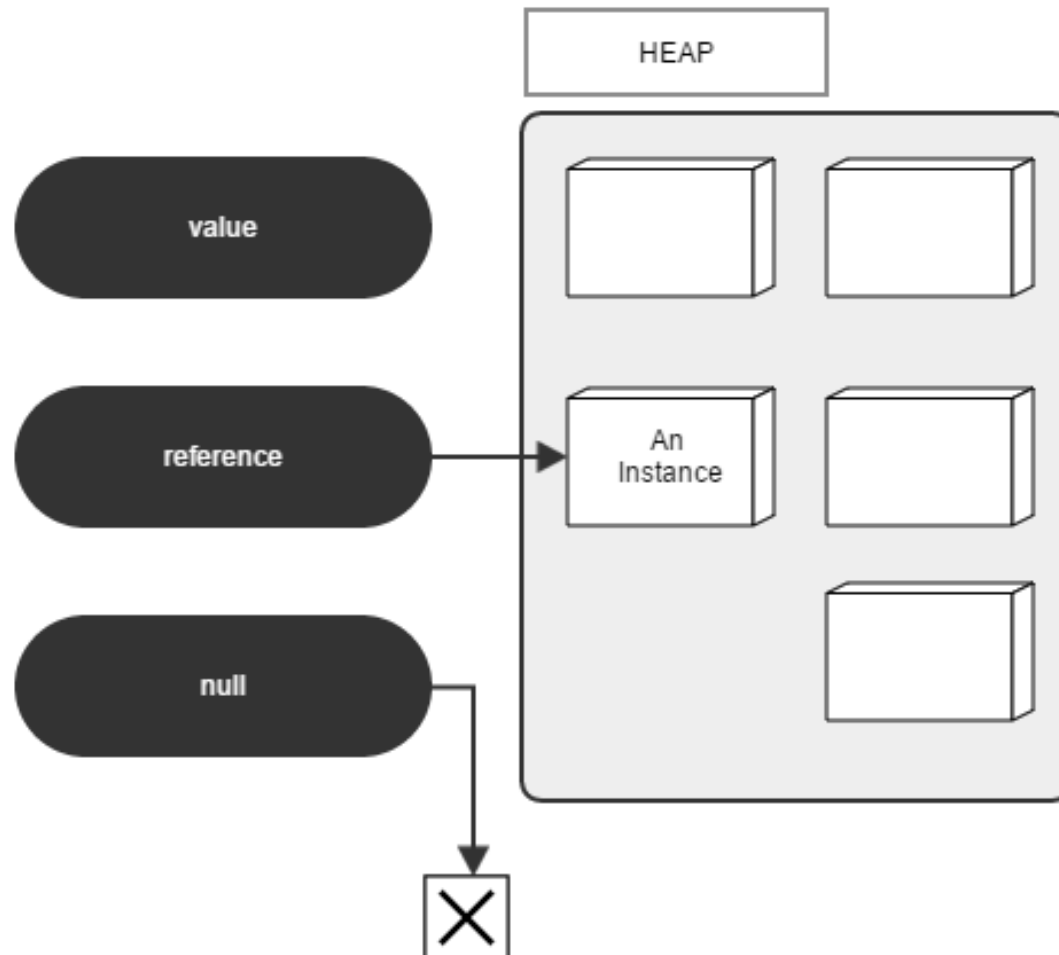
- ✓ **Final variables:** final variables cannot be changed after they have been instantiated.
- ✓ **Final methods:** final methods cannot be overridden in child classes.
- ✓ **Final classes:** classes that cannot be extended; usually marked as such in order to avoid other people breaking highly sensitive functionality.

Class declaration, instantiation, initialization

```
public class Student {  
    // variable  
    public String name = "";  
    public int age = 20;  
    // constructors  
    public Student() { // do nothing };  
    public Student(String studentName) {  
        this.name = studentName;  
    };  
    // methods  
    public void introduceYourself(){  
        System.out.println("Hi, my name is " + this.name);  
    }  
}
```

- ✓ Constructors are used for instantiation, based on their parameters:
 - ✓ Student studentOne= new Student (); // will call the first constructor
 - ✓ Student studentTwo = new Student ("Vlad"); // will call second constructor

References in Java



Object – the Super Class of all Java objects

- ✓ **toString** : returns the representation of the object as an array of characters
- ✓ **equals** : test the equality of two objects of the same type
- ✓ **hashCode** : returns the HASH value (simplified numeric representation of the object) corresponding to the object
- ✓ **getClass** : returns the Class the object has been instantiated with
- ✓ **clone** : creates a shallow copy of the object
- ✓ **finalize** : called by GC before deleting the object

- ✓ This is a language constraint and is not explicit inheritance

OOP – Methods and Parameters

[access modifiers] **ReturnedType** methodName(**ParameterType** param, ... args)

```
public void introduceYourself(){  
    System.out.println("Hi, my name is " + this.name);  
}
```

```
public void sayHello(String personName){  
    System.out.println("Hola "+personName+"!");  
}
```

```
public int getAge(){  
    return this.age;  
}
```

Inheritance Example

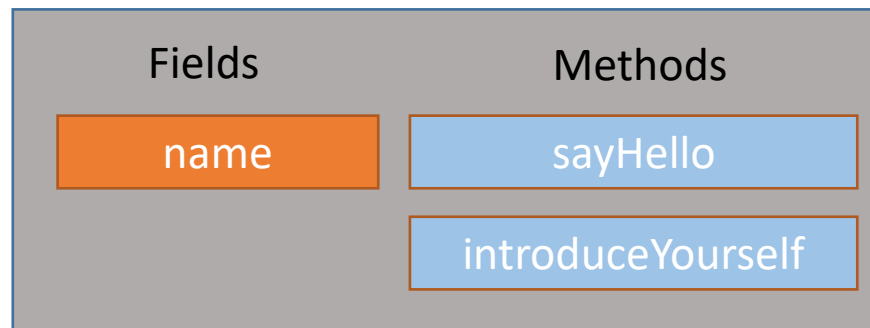
```
public class Person{
    String name = "Vlad";
    public void sayHello(){
        System.out.println("Hello!");
    }
    public void introduceYourself(){
        System.out.println("Hi, my name is " + this.name);
    }
}

public class SpanishStudent extends Person{
    int age = 0;
    public void sayHello(){
        System.out.println("Hola!");
    }
}

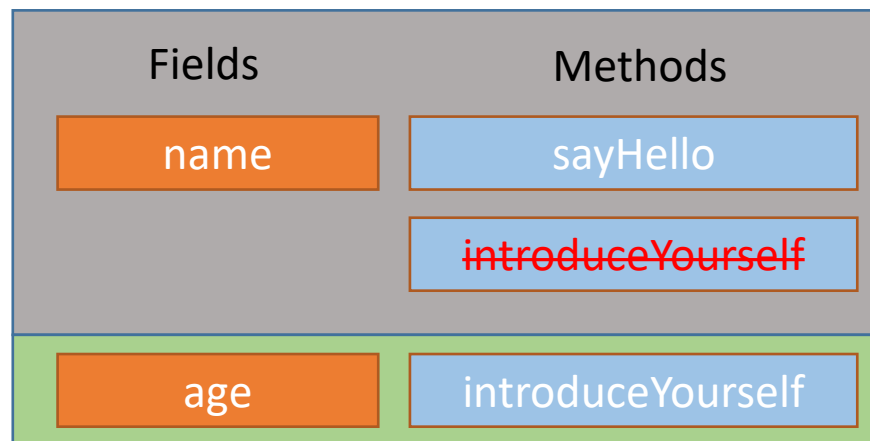
✓ Super classes group common functionality for all subclasses
✓ Sub classes use parent defined behavior and define specific behavior
```

Inheritance Example

Person Class
Memory
Representation



Spanish Student
Class Memory
Representation



Person Class
Fields and Methods

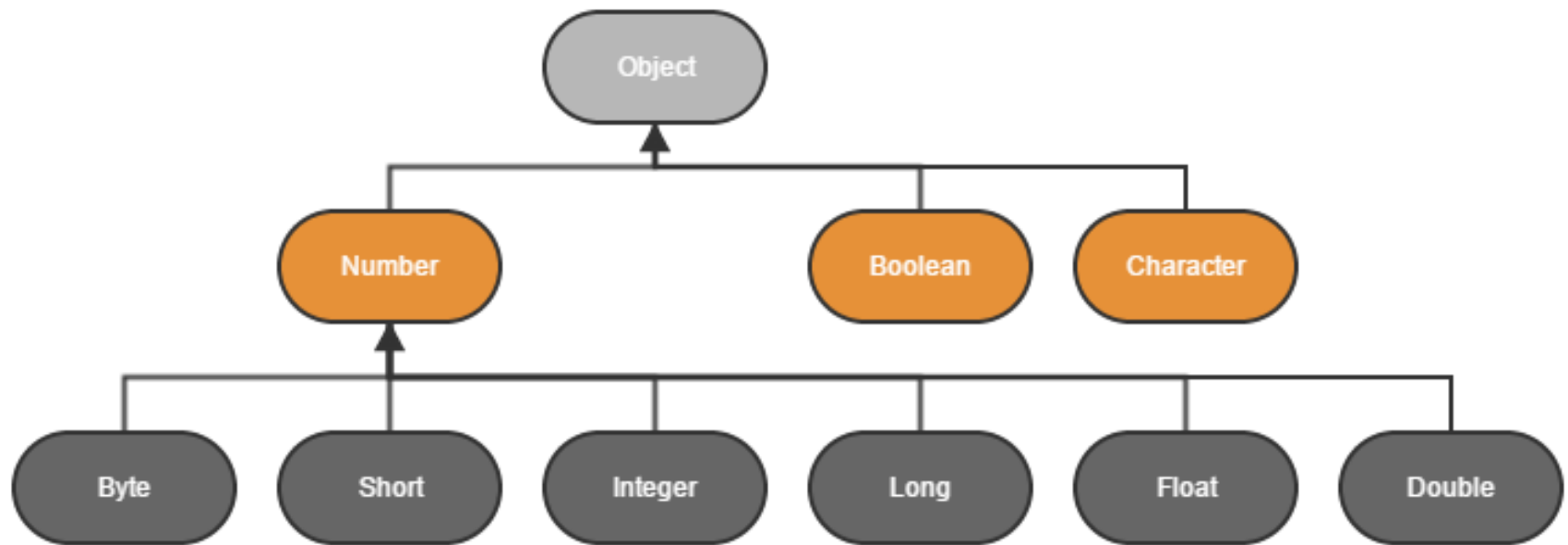


Spanish Student Class
Fields and Methods

OOP – Inheritance and constructors

```
class A {  
    public A() {  
        System.out.println("A");  
    }  
}  
class B extends A {  
    public B() {  
        System.out.println("B");  
    }  
}  
class C extends B {  
    public C() {  
        System.out.println("C");  
    }  
}  
C c = new C();  
// What do you think this will print ?
```

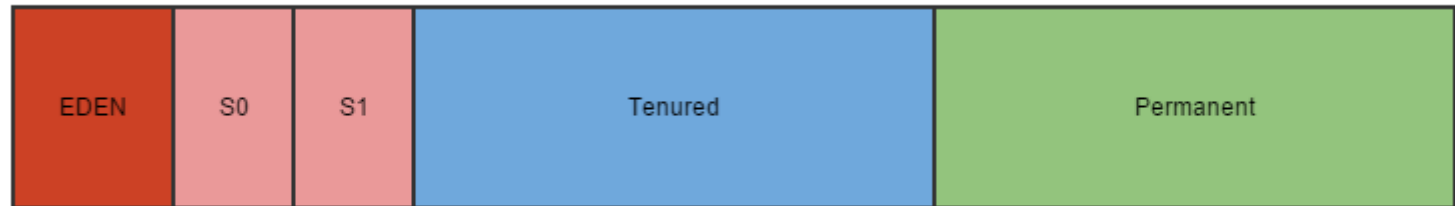
Wrapper Types Use Inheritance



Autoboxing and unboxing

- ✓ **Autoboxing** is the automatic conversion that the Java compiler makes between the primitive types and their corresponding object wrapper classes.
- ✓ Converting an object of a wrapper type to its corresponding primitive value is called **unboxing**. This process takes place when:
 - ✓ A wrapper type is passed as a parameter to a method that expects a value of the corresponding primitive type.
 - ✓ A wrapper type is assigned to a variable of the corresponding primitive type.

Garbage Collection – Heap Structure



Assignments

- ✓ Write a class that represents a person. Extend that class for particular types of people: professor, student, janitor, security guard.
- ✓ **Think about** and write common and specific methods for the above classes.
- ✓ Write a class whose instances represent a single playing card from a deck of cards. Playing cards have two distinguishing properties: rank and suit.
- ✓ Write a class whose instances represent a full deck of cards.
- ✓ Write a small program to test your deck and card classes. The program can be as simple as creating a deck of cards and displaying its cards.

THANK YOU!



Vlad Costel Ungureanu
contact@learnstuffacademy.io

This is a **free** course from LearnStuff.IO
– not for commercial use –