Section 6_OOP Part1 Classes, Constructors

Topics Covered:

Classes

Constructors

IntelliJ short cut

Inheritance

Reference

This vs super

Method overriding vs Overloading

Static vs Instance methods

Static vs Instance variables

Object Class

Classes

Same as C++

How to make classes in IntelliJ

- Make a project as normal
- Drop down menu classes -> src -> right click on package name or on src -> new -> java class

Fields members of a class

Example

```
public class Car
{
    // the car class also inherits some function by default from the base
java class
    private int doors;
    private int wheels;
    private String model;
    private String engine;
    private String colour;
```

```java
    public void setModel(String model){
        String validModel = model.toLowerCase();

        if(validModel.equals("carrera") || validModel.equals("commodore")){
            this.model = model;
        }
        else {
            this.model = "unknown";
        }
    }

    public String getModel(){
        return model;
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        Car porsche = new Car();
        Car holden = new Car();
        porsche.setModel("C");
        System.out.println("The car model is " + porsche.getModel());
    }
}
```

Constructors

Same as C++

NB: some suggest nit calling setters in constructors as it sometimes causes problem (it is a conflicting opinion)

```java
public class Car
{
    // the car class also inherits some function by default from the base
java class
    private int doors;
    private int wheels;
    private String model;
    private String engine;
    private String colour;

    public Car(){
        //calling one constructor from another constructor
        this("porse", "pink");
        System.out.println("Empty Constructor called");
    }

    public Car(String model, String colour){
        this.model = model;
        this.colour = colour;
    }
    public Car(int doors, int wheels, String model, String engine, String 
colour) {
        this.doors = doors;
        this.wheels = wheels;
        this.model = model;
        this.engine = engine;
        this.colour = colour;
    }

    public void setModel(String model){
        String validModel = model.toLowerCase();

        if(validModel.equals("carrera") || validModel.equals("commodore")){
            this.model = model;
        }
        else {
            this.model = "unknown";
        }
    }

    public String getModel(){
        return model;
    }

    public String getColour() {
        return colour;
    }

    public void setColour(String colour){
        this.colour = colour;
    }

}
```

<u>IntelliJ short cut</u>

Code -> generate -> constructor -> (shift and hover mouse to select) -> okay

<u>Inheritance</u>

```
public class Dog extends Animal{
```

The rest is the same as C++

Super calls the constructor that is for the class that we are extending from

Always use super when calling parent methods from child class

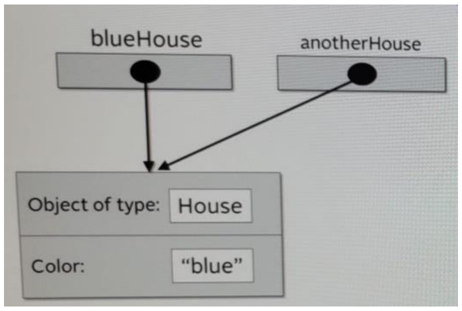
```
public void walk(){
    System.out.println("Dog is walking");
    super.eat();
}
```

<u>Reference</u>

Address of an object in memory

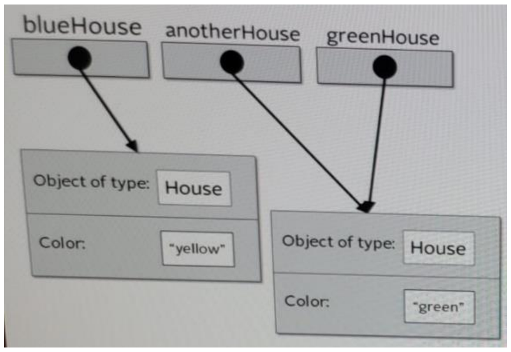
House blueHouse = new House(“blue”);

House anotherHouse = blueHouse;



House greenHouse = new House(“green”);

anotherHouse = greenHouse;



NB: In Java you always have reference to an object in memory, there is no way to access an object directly everything is done using reference.

<u>This vs super</u>

<u>Super</u> it is used to access or call parent class members (variables and methods)

<u>This</u> it is used to call the current class members (variables and methods).

This is required when we have a parameter with the same name as an instance variable (field)

```
public Car(){
    //calling one constructor from another constructor
    this("porse", "pink");
    System.out.println("Empty Constructor called");
}

public Car(String model, String colour){
    this.model = model;
    this.colour = colour;
}
```

Nb: we can use both anywhere in a class except static areas (the static block or static method). Any attempt to do so will lead to compile-time errors (more on static later in the course)

Method overriding vs Overloading

Overloading means providing two or more separate methods in a class with the same name but different parameters. The return parameter may or may not be different and that allows us to reuse the same method name.

It is also known as compile time polymorphism

Overriding means defining a method in a child class that already exists in the parents' class with the same signature(same name, same arguments)

It is also known as run time polymorphism and dynamic method dispatch because the method that is going to be called is decided at run time by the Java virtual machine

When we override it is recommended to put **@Override** immediately above the method definition. This is an annotation that the compile reads and will then show us an error if we do not follow overriding rules correctly.

```
@Override
public void eat() {
    System.out.println("Dog.eat() is called");
    chew();
    // calls the eat in Animal (parent class)
    super.eat();
}
```

Only inherited methods can be overwritten meaning that parent class methods can be overwritten in child classes

Constructors and private methods can not be overridden

Static vs Instance methods

Static methods are declared using a static modifier

In static methods we can't use this keyword

Main is a static method, and it is called by the Java virtual machine when it starts and application.

```java
class Calculator {
    public static void printSum(int a , int b){
        System.out.println("sum " + (a + b));
    }
}

public class Main{
    public static void main (String[] args){
        Calculator.printSum(5,10);
        printHello(); // shorter form of Main.printHello();
    }

    public static void printHello(){
        System.out.println("Hello");
    }
}
```

Instance methods belong to an instance of a class

To use an instance method, we have to instantiate the class first usually by using the new keyword.

Instance methods can access instance methods and instance variables directly

Instance methods can also access static methods and static variables directly

```java
class Dog{
    public void bark(){
        System.out.println("woof");
    }
}

public class Main{
    public static void main(String[] args){
        Dog rex = new Dog();  // create instance
        rex.bark();  // call instance method
    }
}
```

Static vs Instance variables

Static variables

Declared using static keyword

Static variables are also known as static member variables

Every instance of a class shares the same static variable.

For example, when reading user input using scanner, we can declare scanner as a static variable that way static methods can access it directly

```java
class Dog{
    private static String name;

    public Dog(String name){
        this.name = name;
    }
    public void printName(){
        System.out.println("name = " + name);
    }
}

public class Main{
    public static void main(String[] args){
        Dog rex = new Dog("rex");  // create instance (rex)
        Dog fluffy = new Dog("fluffy");  // create instance (fluffy)
        rex.printName(); // prints fluffy
        fluffy.printName(); // prints fluffy
    }
}
```

Instance variables

They don't use static keyword

They are also known as fields or member variables

They belong to an instance of a class

```java
class Dog{
    private String name;

    public Dog(String name){
        this.name = name;
    }
    public void printName(){
        System.out.println("name = " + name);
    }
}

public class Main{
    public static void main(String[] args){
        Dog rex = new Dog("rex");  // create instance (rex)
        Dog fluffy = new Dog("fluffy");  // create instance (fluffy)
        rex.printName(); // prints rex
        fluffy.printName(); // prints fluffy
    }
}
```

Object Class

Every Java classs inherits from the class Object

Class object is the root of the class hierarchy. Every class has object as the super class. All objects including arrays implement the methods of this class.

Even though you don put extends after a class name, every class automatically inherits from the Object class (that's just how java is)

public class Main extends Object {

}