Object Oriented Programming in Java

Course Outline

1. Introduction to Java
2. Data Handling and Functions
3. **Object Oriented Programming in Java**
4. Packages and Multithreading
5. Collections

Objectives

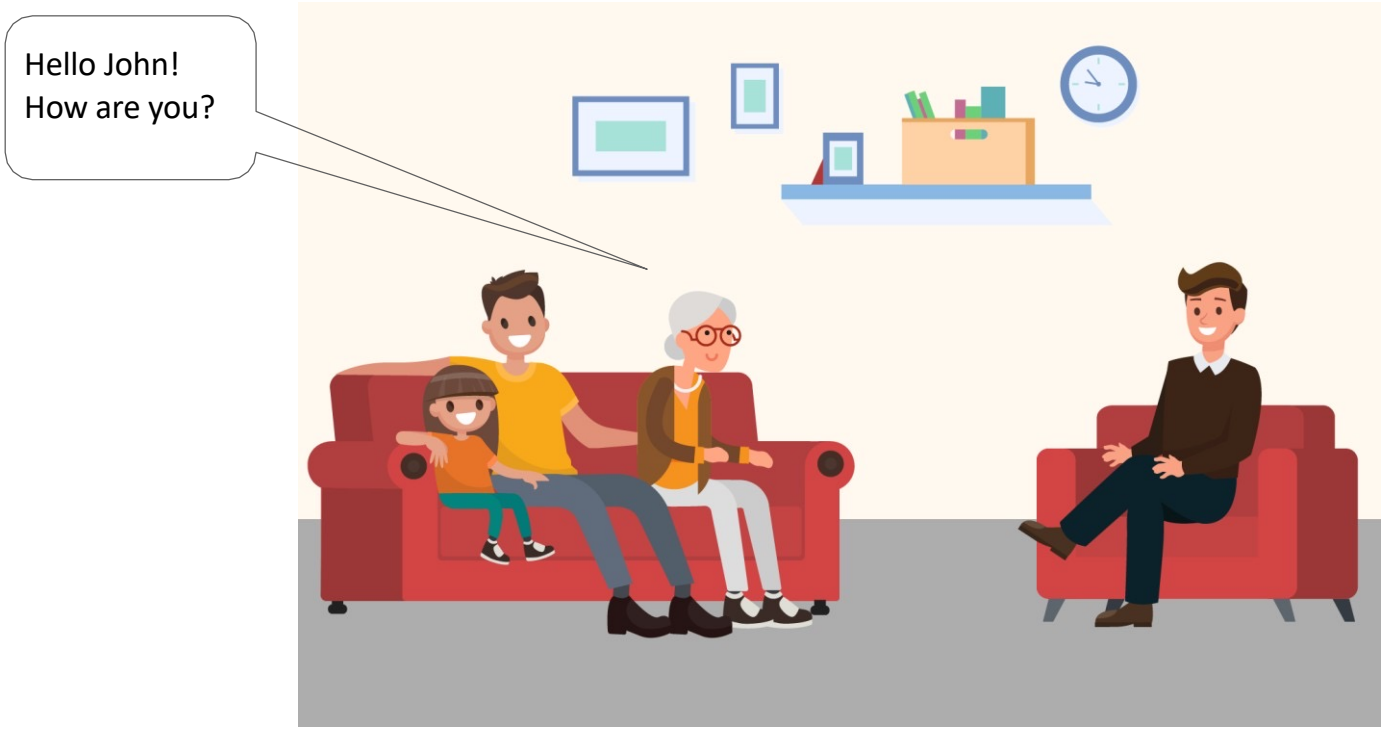
After completing this module, you should be able to:

- Implement classes and objects in Java
- Create class constructors
- Overload constructors
- Inherit classes and create sub-classes
- Implement abstract classes and methods
- Use static keyword

John has Visitors!

John’s Aunt and Cousin came to meet him.

John has Visitors!



John has Visitors!

John has Visitors!

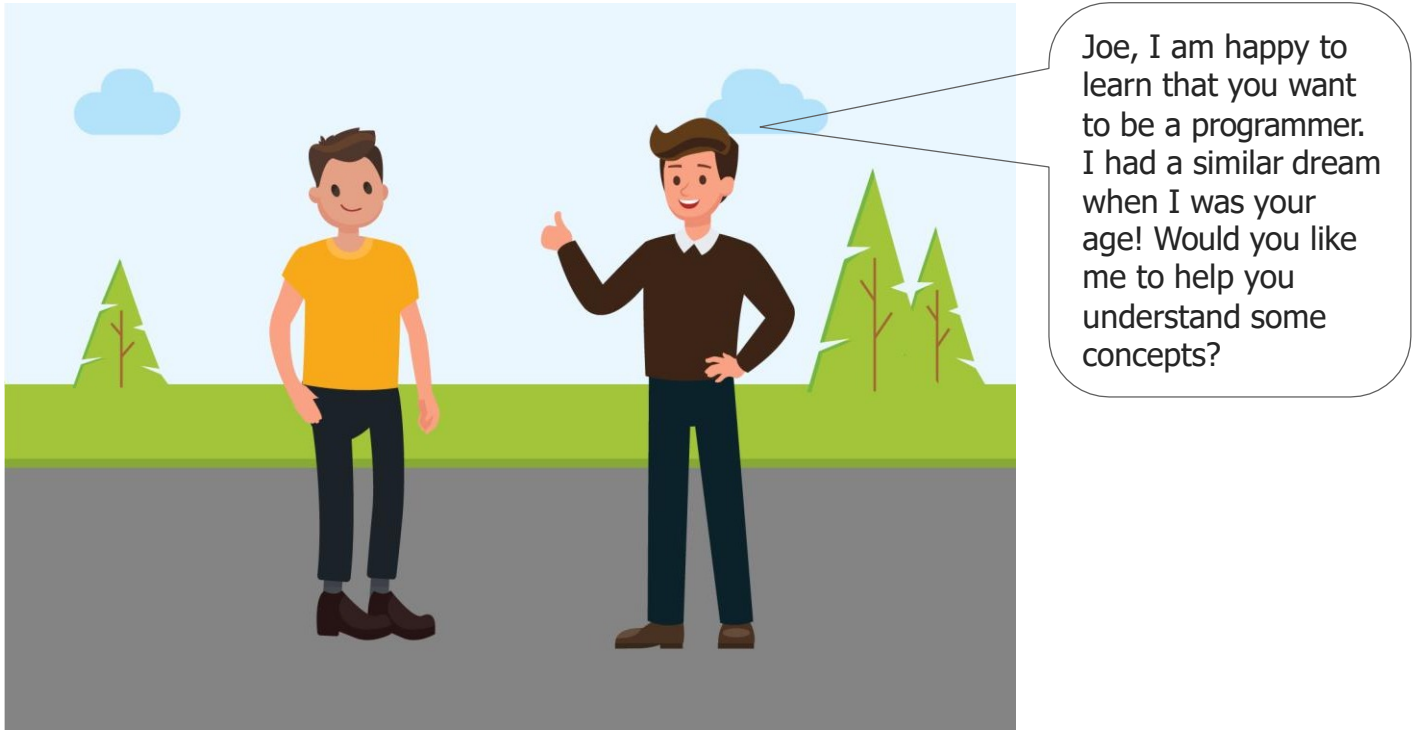
We have come to congratulate you on your promotion. I am happy to see you become a good programmer!

John has Visitors!

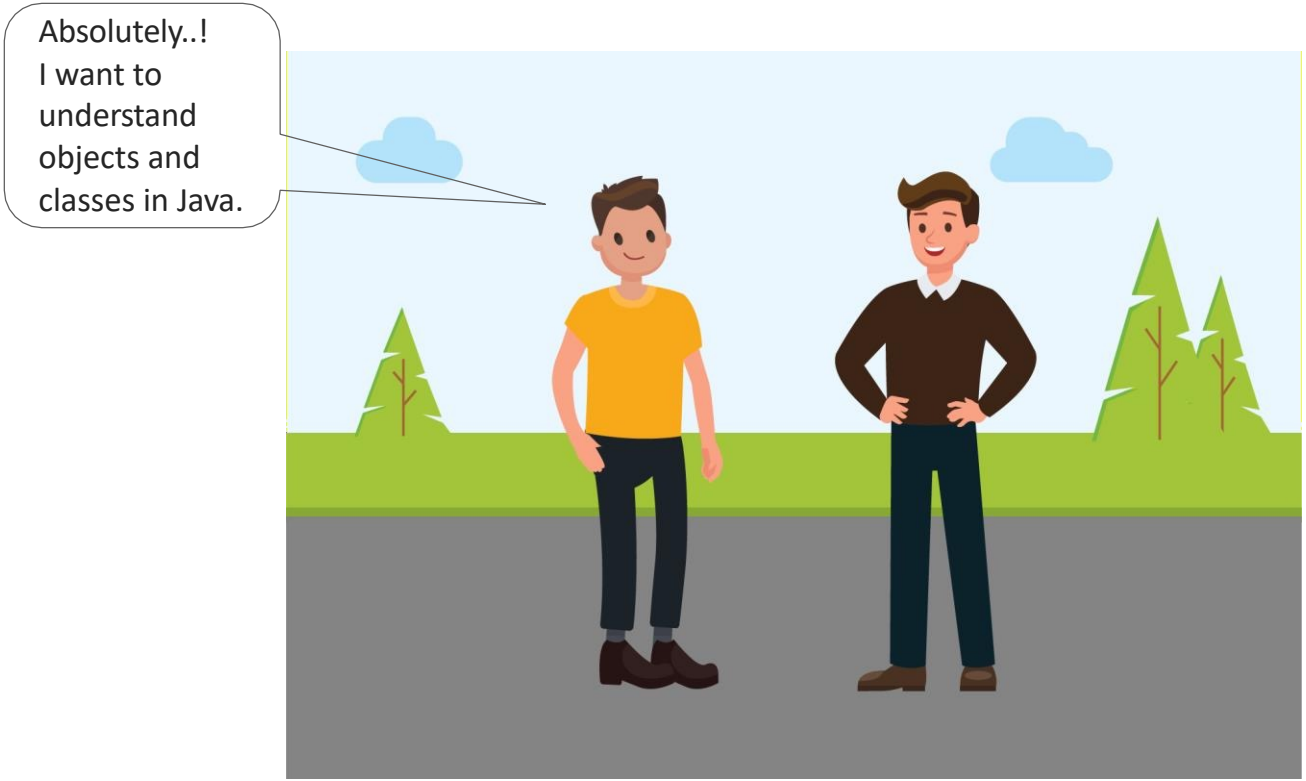
You will be happy to know that Joe has enrolled for Computer science. See if you can guide him to become like you.

John has Visitors!

John helps Joe!



Joe’s Concern



John explains Objects

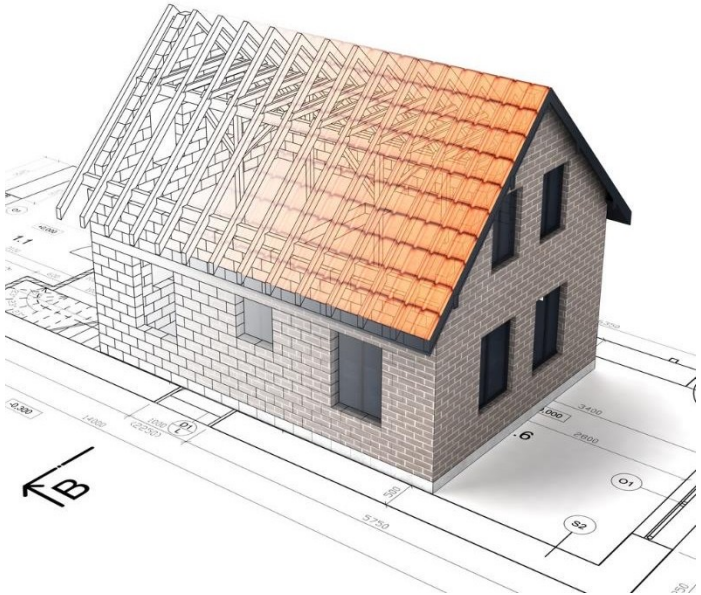
Joe, Look around you! You see leaves, trees, grass, door etc., everything around you is an object. Everything that has a state and behavior is object. The tree there is at rest. It is green and its trunk is thick, which is its behavior. Even you and me are objects.

John explains Classes

Classes are a blue print by which we create objects. For example, we are objects (john, joe etc.) under the class (humans).

Classes and Objects

Let us consider one more example:

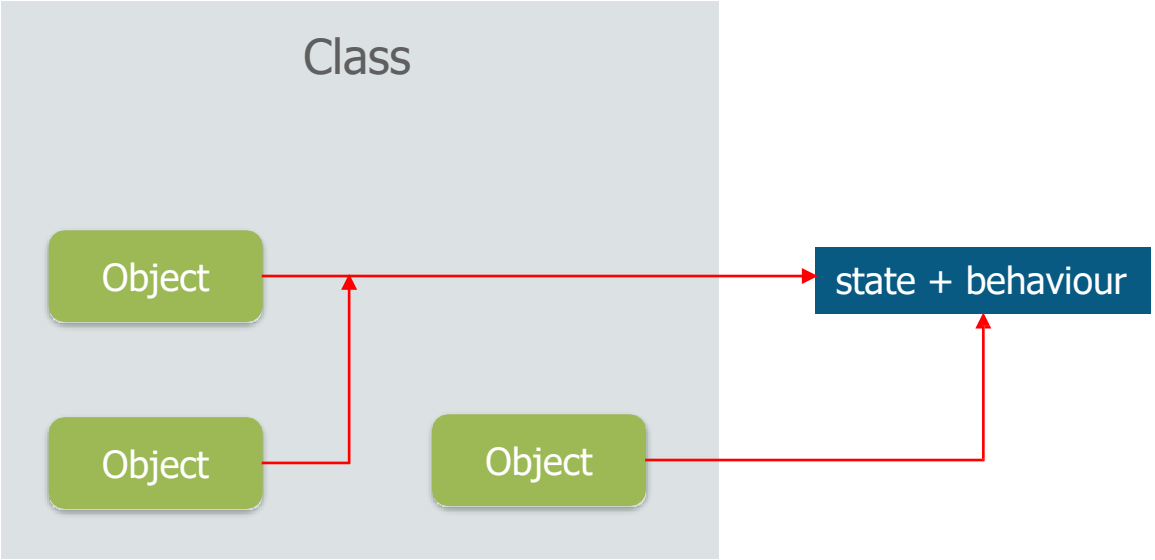


- When we construct a house, we create a plan. This is called the blueprint of the house. The house is build, based on the plan

- Classes and objects are similar to this. Classes are like plan and objects are similar to the constructed house

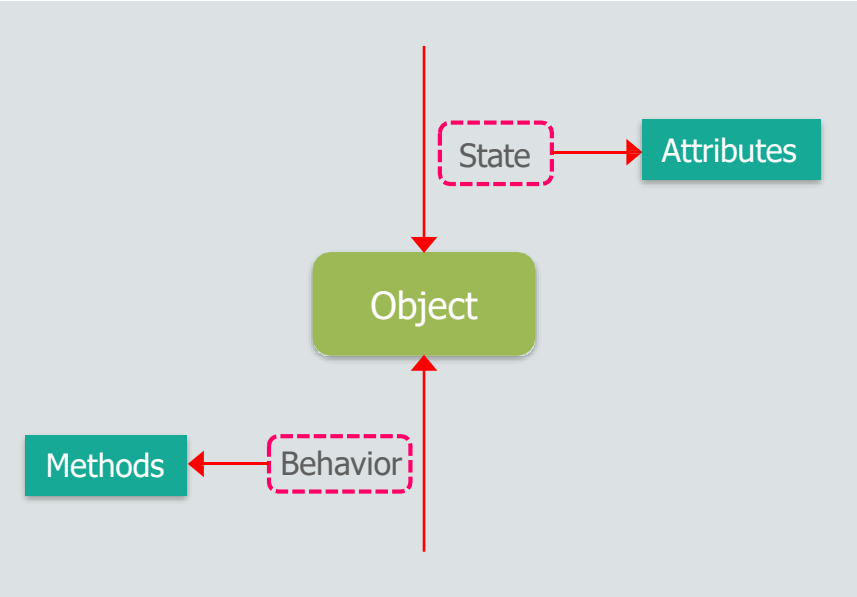
Classes and Objects (contd.)

- A class is the blueprint from which specific objects are created
- Anything that has a state and behavior is **object**



Attributes and Methods

- Attributes are state of an object
- Methods are behaviour of an object



OOPS – Use Cases

Use cases (Where), *Problem:, *How it can be resolved.

A software development company develops a project for Insurance company. After completing the design and during the mid phase of the coding, insurance company gives changes in the requirements. Since software development firm is using System Analysis and Design Software engineering process, it was difficult to handle the changes. Later on software company implemented the code using Object oriented programming, hence they are able to make the changes easily

OOPS – Use Cases

Use cases (Where), *Problem:, *How it can be resolved.

A software development company develops a project for Insurance company. After completing the design and during the mid phase of the coding, insurance company gives changes in the requirements. Since software development firm is using System Analysis and Design Software engineering process, it was difficult to handle the changes. Later on software company implemented the code using Object oriented programming, hence they are able to make the changes easily

- Object Oriented Programming is based on objects or real time entity or real world entities. Object has attributes and methods. Any new method can be added or deleted easily in an object / a class just by adding it/deleting it. Hence, Object oriented program can accommodate changes given by the Insurance company easily

Classes in Java

- A Class in Java is defined using the keyword "**class**".

For example:

```
public class student {
    int reg_no;
    String name;
    String stream;

    void Study() {
    }

    void WriteExams() {
    }

    void AttendClasses() {
    }

    void WriteAssignments() {
    }
}
```

Classes in Java

- A Class in Java is defined using the keyword "**class**".

For example:

```
public class student {
    int reg_no;
    String name;
    String stream;

    void Study() {
    }

    void WriteExams() {
    }

    void AttendClasses() {
    }

    void WriteAssignments() {
    }
}
```

Objects in Java

- Objects to a class is created by using the keyword “new”. new allocates memory for the object

For example:

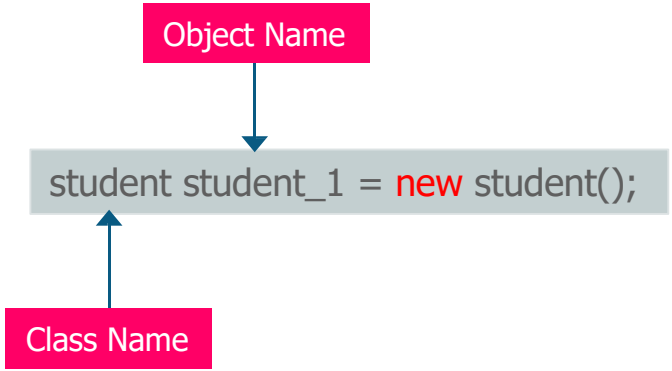
```
student student_1 = new student();
```

Class Name

Objects in Java

- Objects to a class is created by using the keyword “new”. new allocates memory for the object

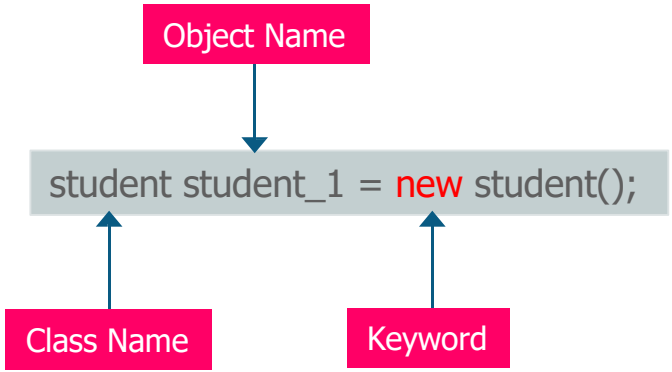
For example:



Objects in Java

- Objects to a class is created by using the keyword “new”. new allocates memory for the object

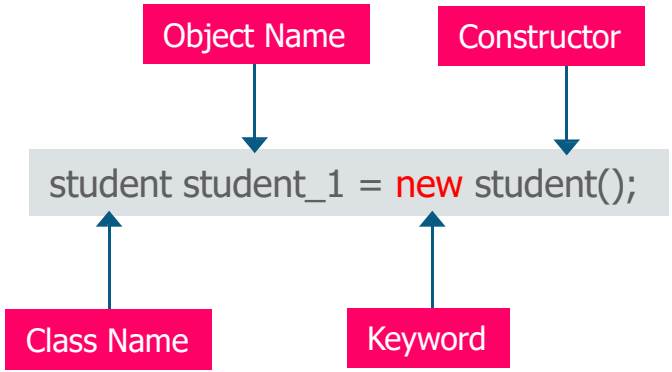
For example:



Objects in Java

- Objects to a class is created by using the keyword “new”. new allocates memory for the object

For example:

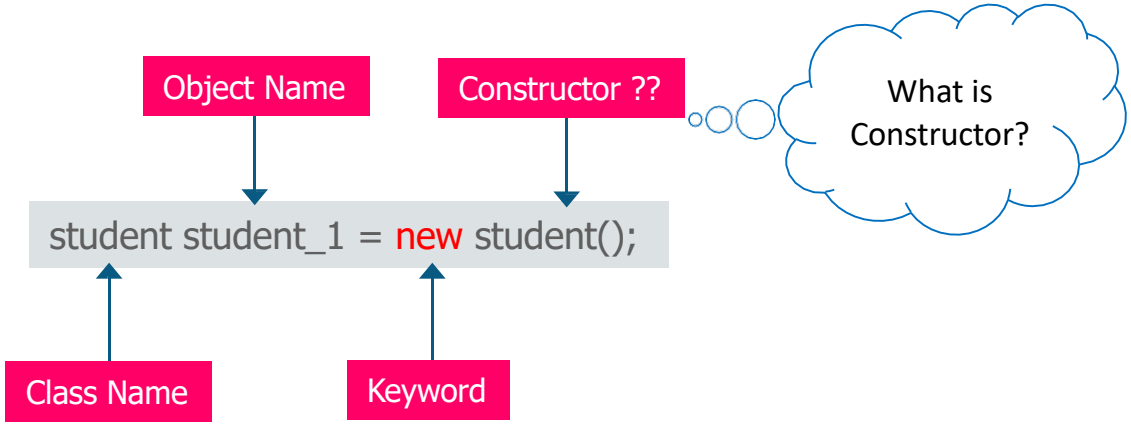


- Here student_1 is the object of the class student
- With student_1 any of the method of the class can be invoked

Objects in Java

- Objects to a class is created by using the keyword “new”. new allocates memory for the object

For example:



- Here student_1 is the object of the class student
- With student_1 any of the method of the class can be invoked

Why do we use Constructors?

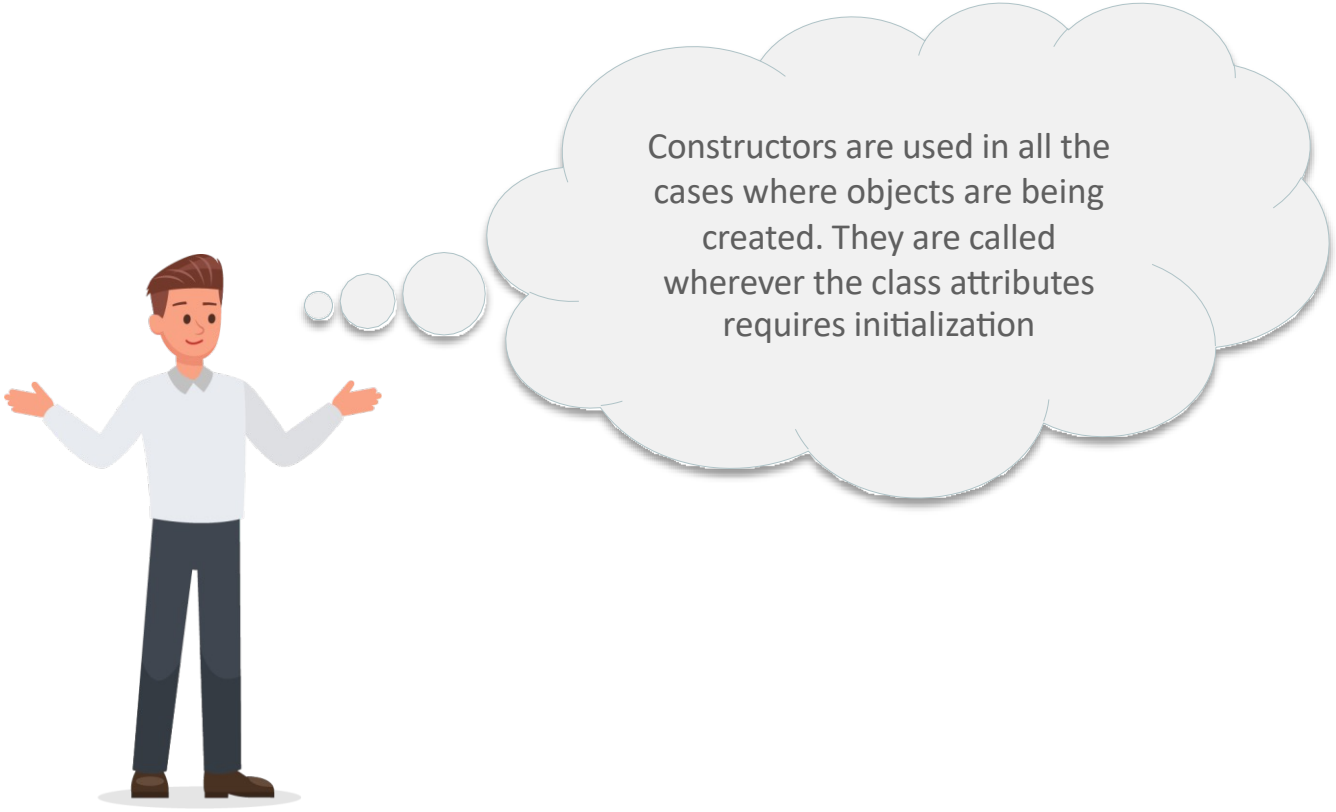
Why do we use Constructors?

Classes will have attributes. Before using them, if they need to be initialized then it has to be placed in a method and this method has to be called every time the object is being created.

To avoid this process, a special function by the same name of the class is created for initializing.

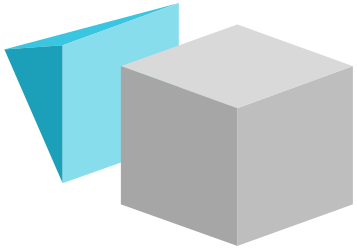
Where do we use Constructors?

Where do we use Constructors?



Constructors

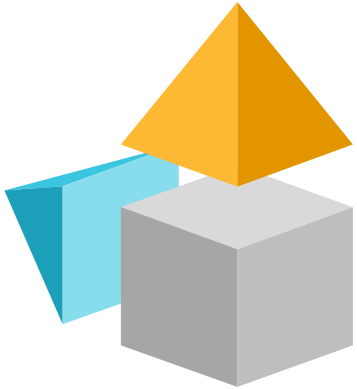
A constructor resembles an instance method, but it differs from a method in that it has no explicit return type



Constructors

Used in the creation of an object

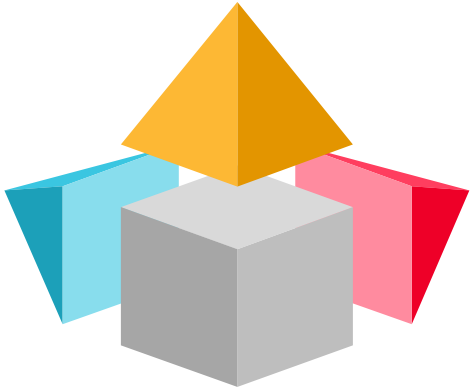
A constructor resembles an instance method, but it differs from a method in that it has no explicit return type



Constructors

Used in the creation of an object

A constructor resembles an instance method, but it differs from a method in that it has no explicit return type

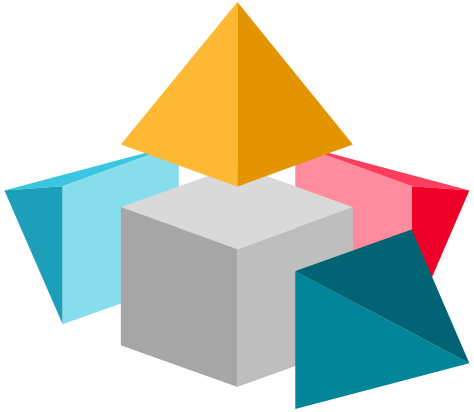


A Special method with no return type

Constructors

Used in the creation of an object

A constructor resembles an instance method, but it differs from a method in that it has no explicit return type



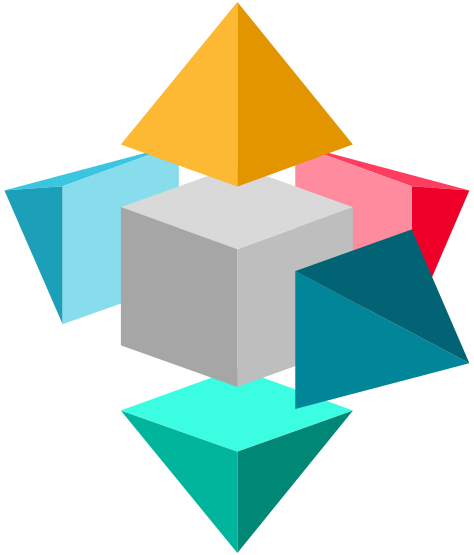
A Special method with no return type

Must have the same name as the class it is in

Constructors

Used in the creation of an object

A constructor resembles an instance method, but it differs from a method in that it has no explicit return type



A Special method with no return type

Must have the same name as the class it is in

Used to initialize the object

Constructors

Used in the creation of an object

A constructor resembles an instance method, but it differs from a method in that it has no explicit return type

A Special method with no return type

If not defined, will initialize variables to default value.

Must have the same name as the class it is in

Used to initialize the object

Constructors

Used in the creation of an object

A constructor resembles an instance method, but it differs from a method in that it has no explicit return type

A Special method with no return type

If not defined, will initialize variables to default value.

Must have the same name as the class it is in

Used to initialize the object

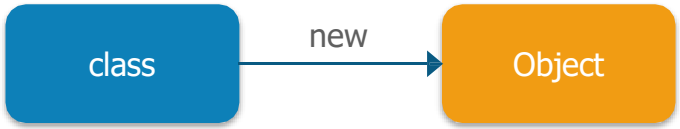
Incase a constructor is not defined then a **default constructor** is called which initializes the instance variables to default value

How do Constructors work?

- The moment object of a class is created, constructor of the class is called which initializes the class attributes

How do Constructors work?

- The moment object of a class is created, constructor of the class is called which initializes the class attributes



How do Constructors work?

- The moment object of a class is created, constructor of the class is called which initializes the class attributes

In Class Question

1. What is the difference between a regular function and a constructor?

In Class Question - Solution

1. What is the difference between a regular function and a constructor?

Solution: Constructor will not return any value a regular function can. Constructor will have the same name as the class but not the regular functions.

Example of Constructor

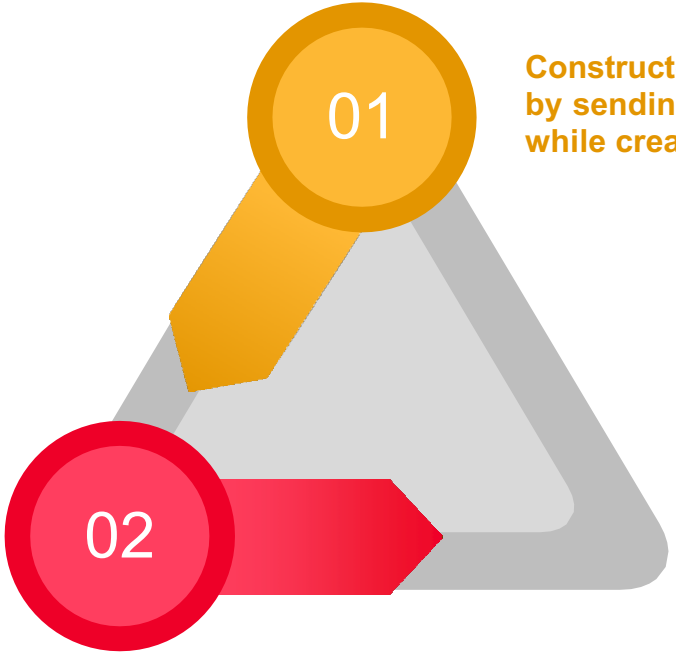
```java
public class student {
    int reg_no;
    String name;
    String stream;

    // Constructor.
    student() {
        reg_no = 0;
        name = "";
        stream = "";

        System.out.println("In the constructor of student..");
    }

    public static void main(String args[]) {
        student student_1 = new student();
    }

  }
```

When **student_1** object is created, **student()** constructor will be executed

Constructor with Parameters

Constructors can be called by sending parameters too while creating the object

Constructor with Parameters

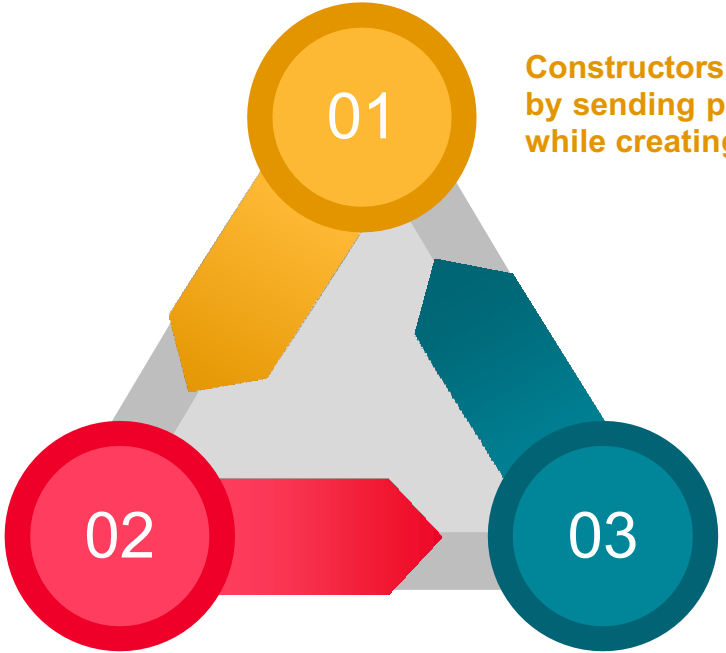


Constructors can be called by sending parameters too while creating the object

For example, student information can be passed on while creating the student object

```
student student_1 = new student(10, "Rohit", "CS") ;
```

Constructor with Parameters



Constructors can be called by sending parameters too while creating the object

For example, student information can be passed on while creating the student object

Here student_1 is created by passing the student object details. Lets see how it is done!

```
student student_1 = new student(10, "Rohit", "CS") ;
```

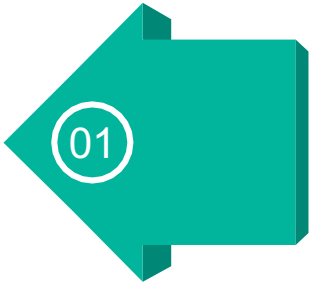
Example for Constructors with Parameters

```java
public class student {
    int reg_no;
    String name;
    String stream;

    // Constructor.
    student(int reg, String name1, String stream1) {
        reg_no = reg;
        name = name1;
        stream = stream1;
        System.out.println("In the constructor with arguments of student..");
    }

    public static void main(String args[]) {
        student student_1 = new student(10, "Rohit", "CS");
    }
}
```

Parameterized Constructor

What is Constructor Overloading & why do we use it?

Constructor Overloading is not very different from method overloading



What is Constructor Overloading & why do we use it?

Constructor Overloading is not very different from method overloading

01

02

Constructor overloading means that you have multiple constructors with same name but different signature

What is Constructor Overloading & why do we use it?

Constructor Overloading is not very different from method overloading

01

02

Constructor overloading means that you have multiple constructors with same name but different signature

Why do we use Constructor Overloading?

- We have different set of data which needs to be assigned or initialized while creating the object. With default constructor, it is not possible. Hence constructor overloading has to be used for initializing the object with various kinds/formats of data. Hence it provides flexibility

Where do we use Constructor Overloading?

- Constructor Overloading is used in various places in Java

For example: Integer datatype → Integer has 2 constructors

```
public class ConstructorOverloading {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        Integer var1 = new Integer(5);

        Integer var2 = new Integer("5");

        System.out.println(var1);
        System.out.println(var2);

    }
```

Integer(int value)

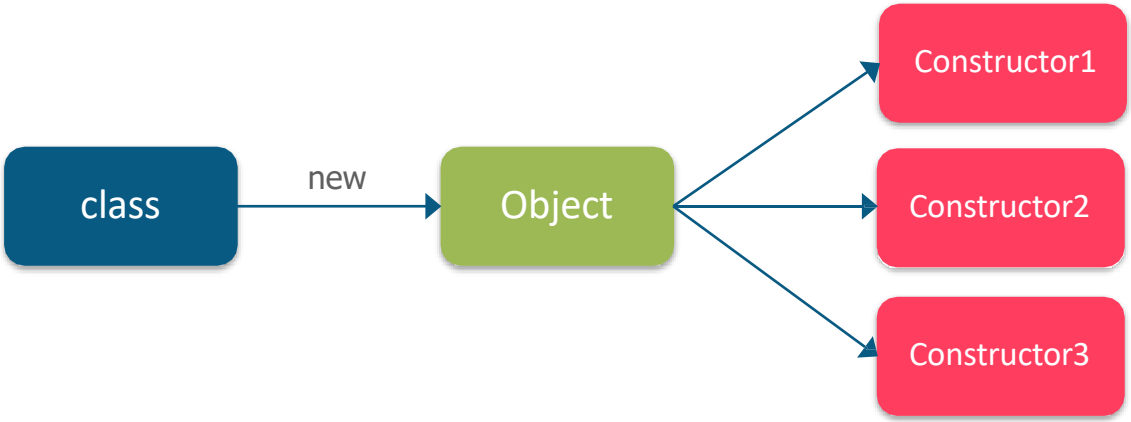
Integer(String s)

Overloaded Constructors

These overloaded constructors allows flexibility

How Constructor Overloading works?

- The moment object of a class is created, constructor of the class is called which initializes the class attributes
- The constructor that has been called will be used to initialize the object



Inheritance

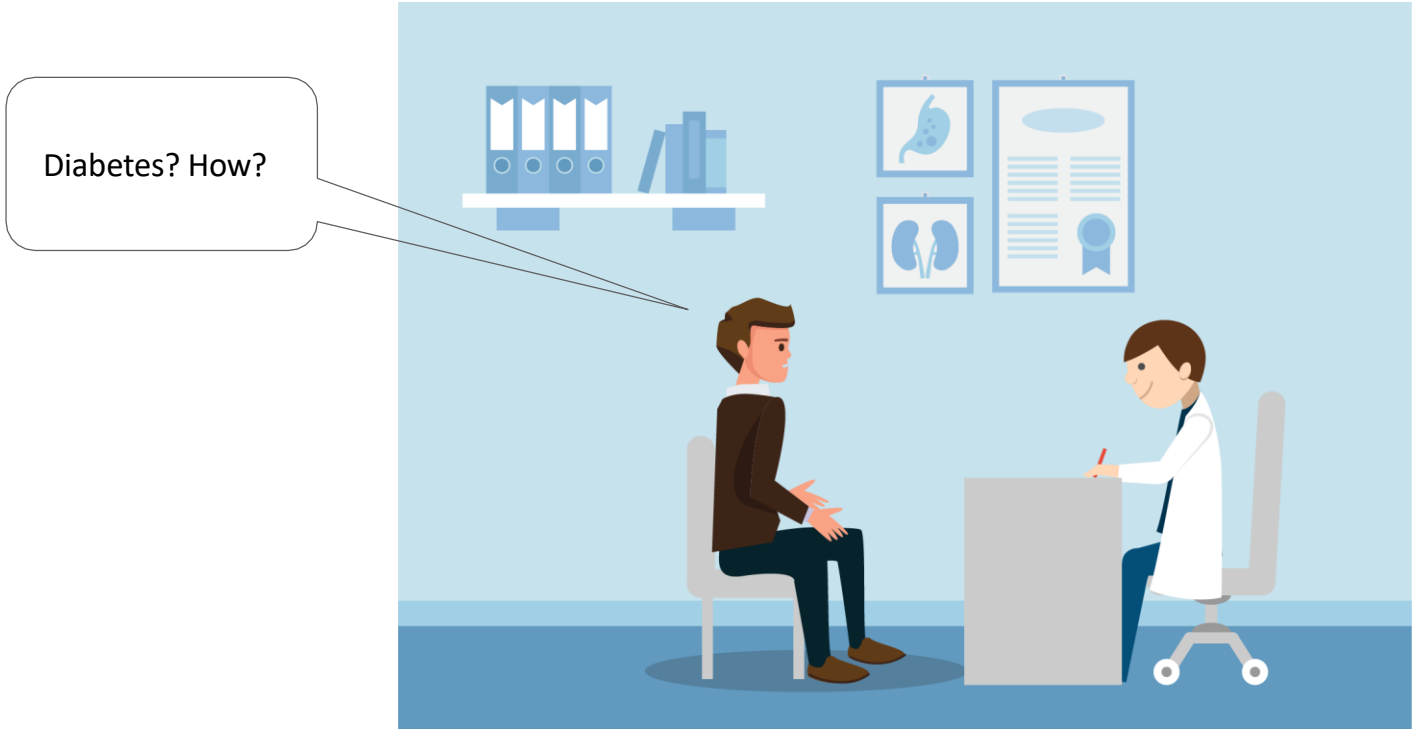
John Visits a Doctor

Hello Doctor!
I am not keeping well for some time. Could you please check what's wrong?

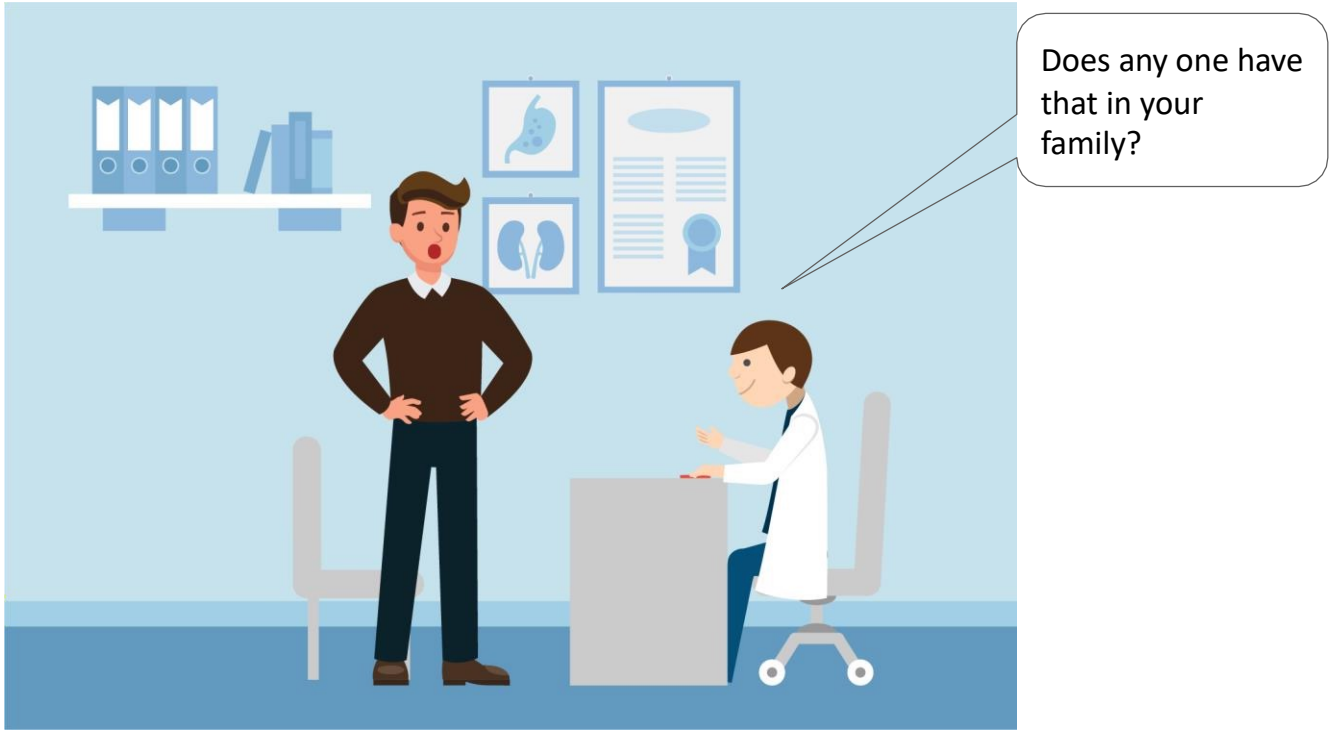
John is not well..

John has diabetes!

John is sad!



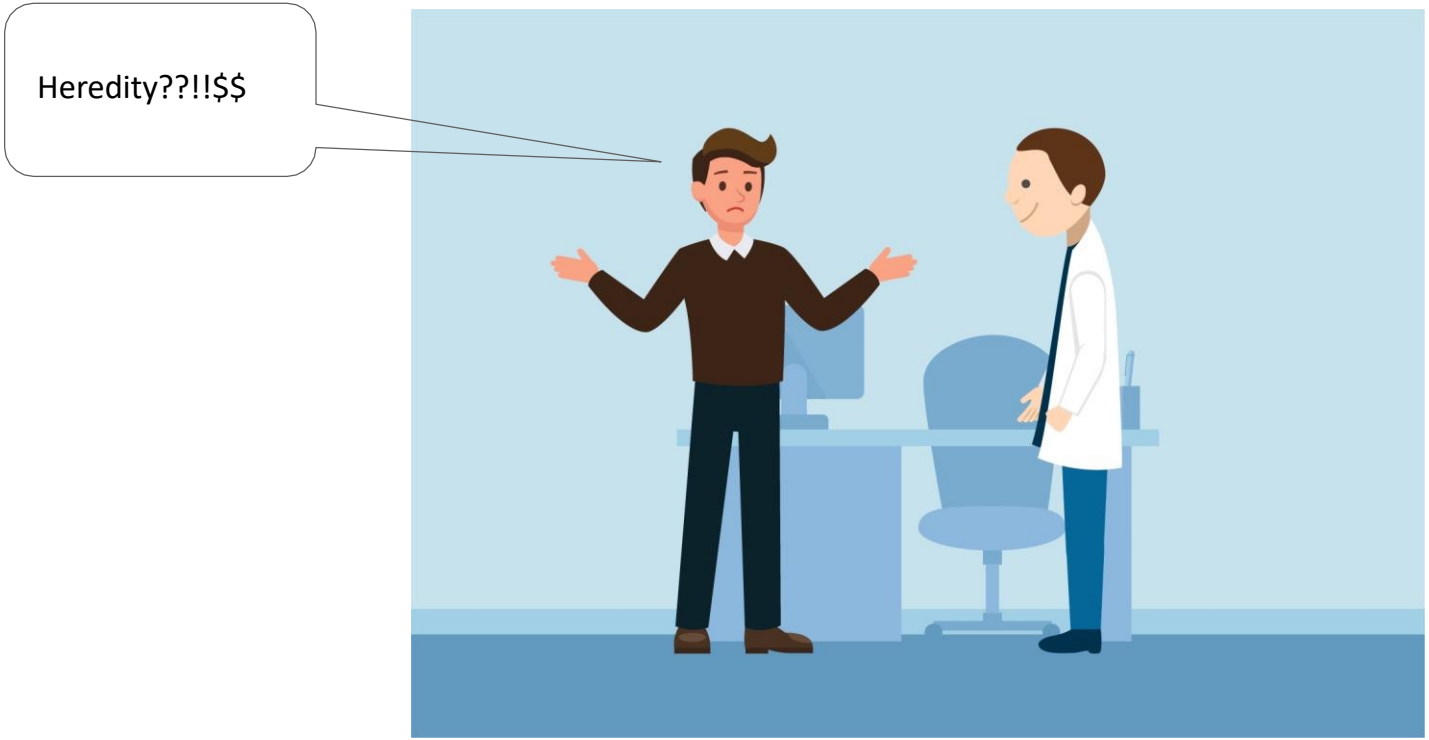
It’s Heredity…



It’s Heredity…

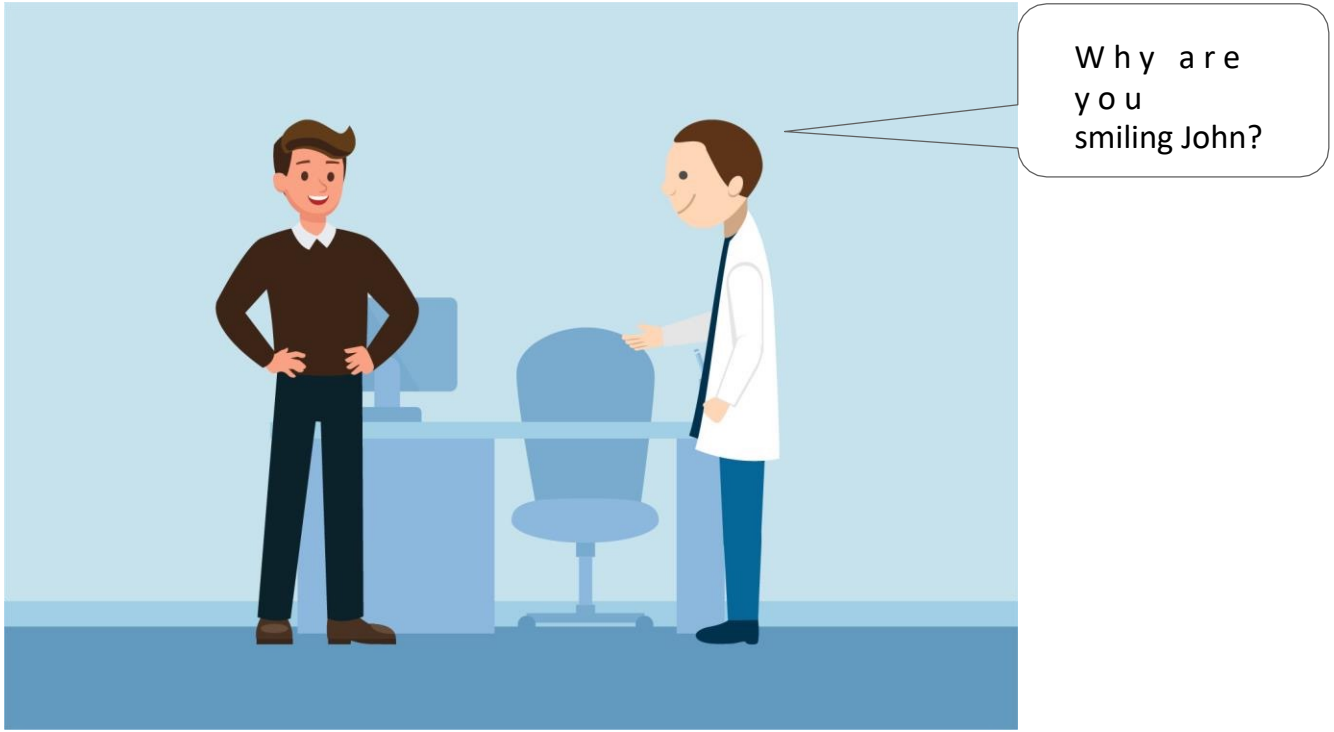
It’s Heredity…

John Wonders How?

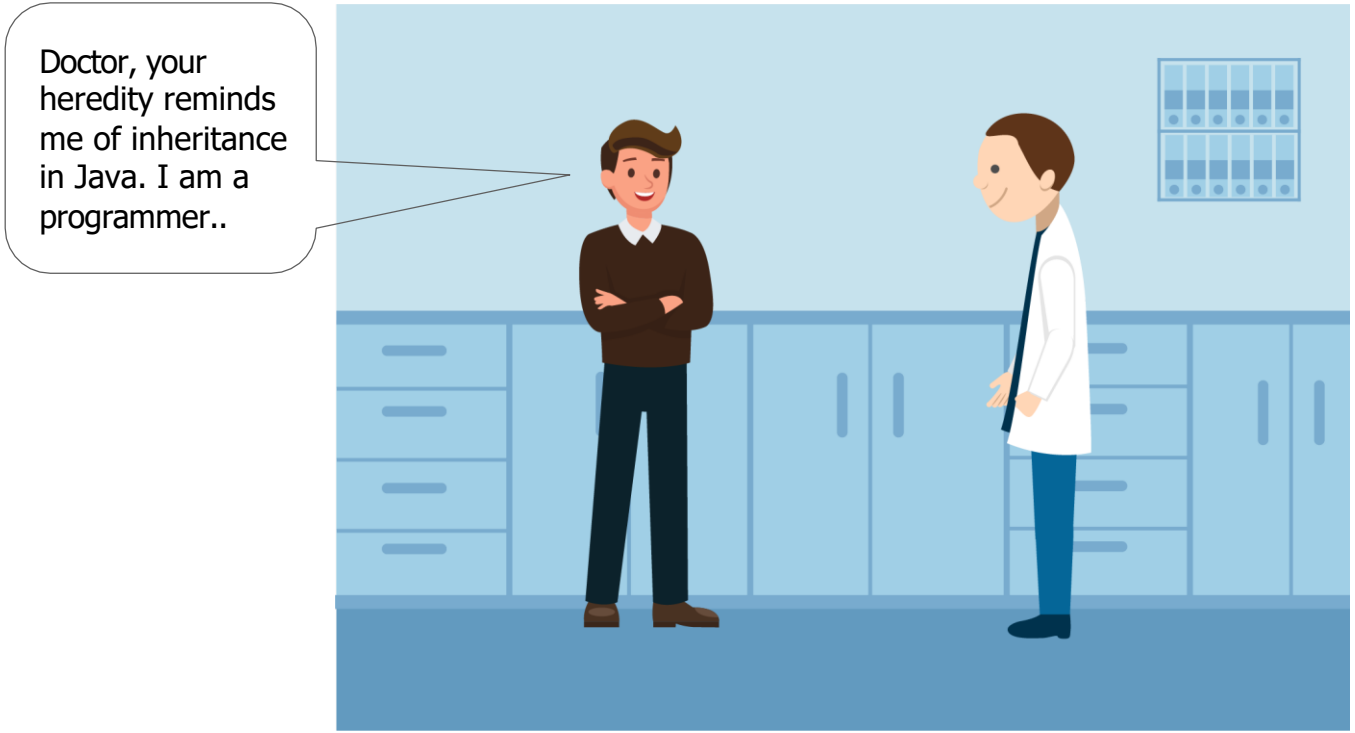


John gets an Answer!

John is Smiling..



Inheritance Vs Heredity



Inheritance Vs Heredity

Let’s Learn Inheritance!

Why do we use Inheritance?

Why do we use Inheritance?

Since some common properties and methods are required by few classes hence they have to be implemented in all those classes.

To avoid code redundancy, a class is developed with the common attributes and methods and it is used as a base class for the derived classes.

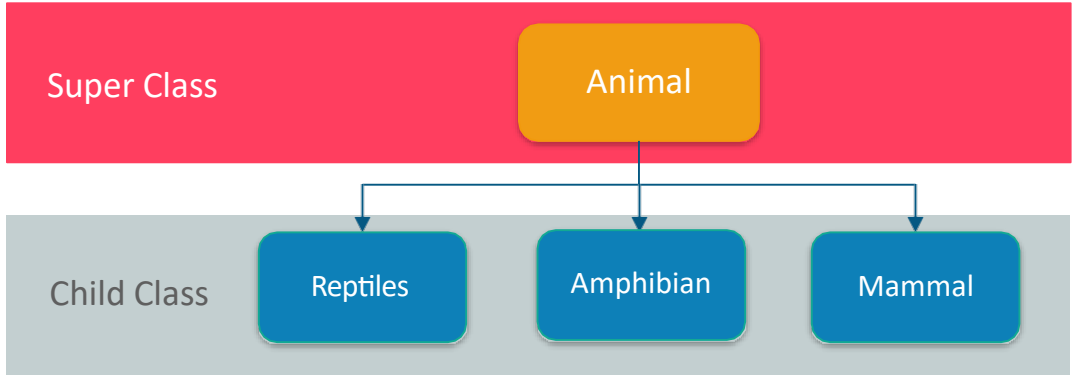
Where do we use Inheritance?

Where do we use Inheritance?

Inheritance

- The child classes inherits all the attributes of the parent class
- They also have their distinctive attributes

For Example:

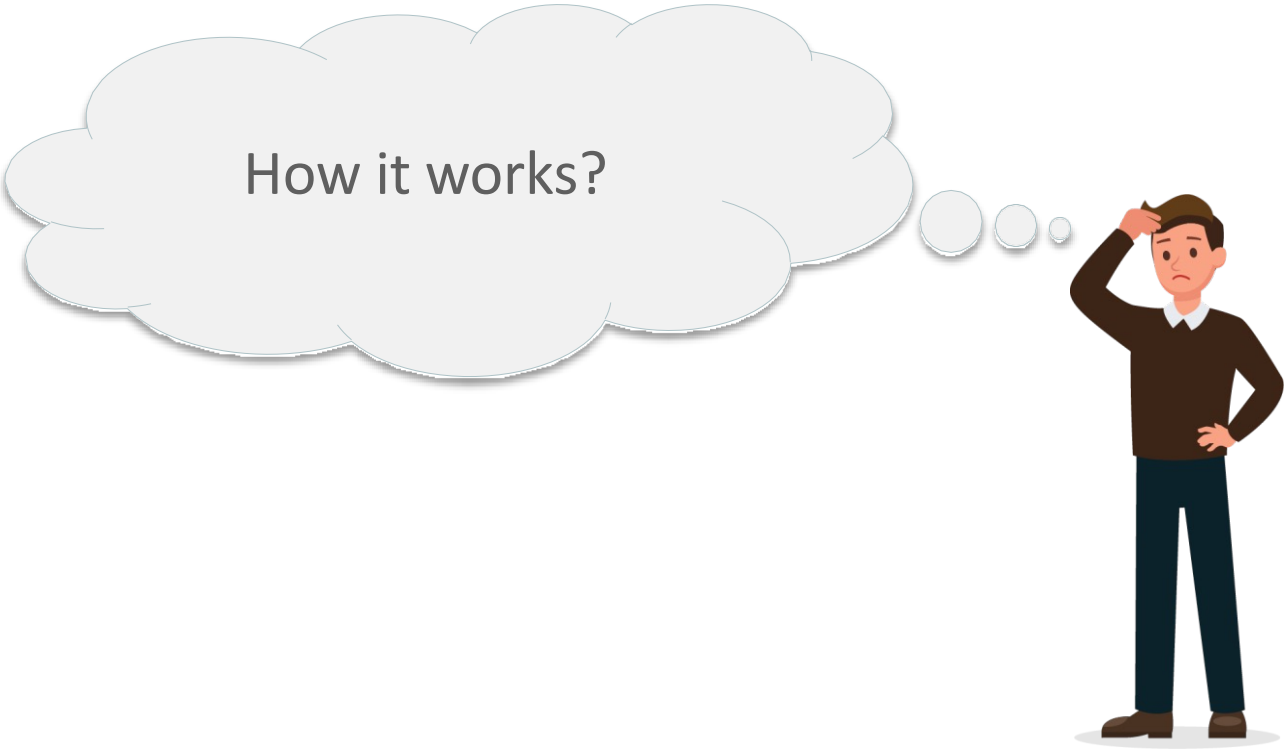


Inheritance - Example

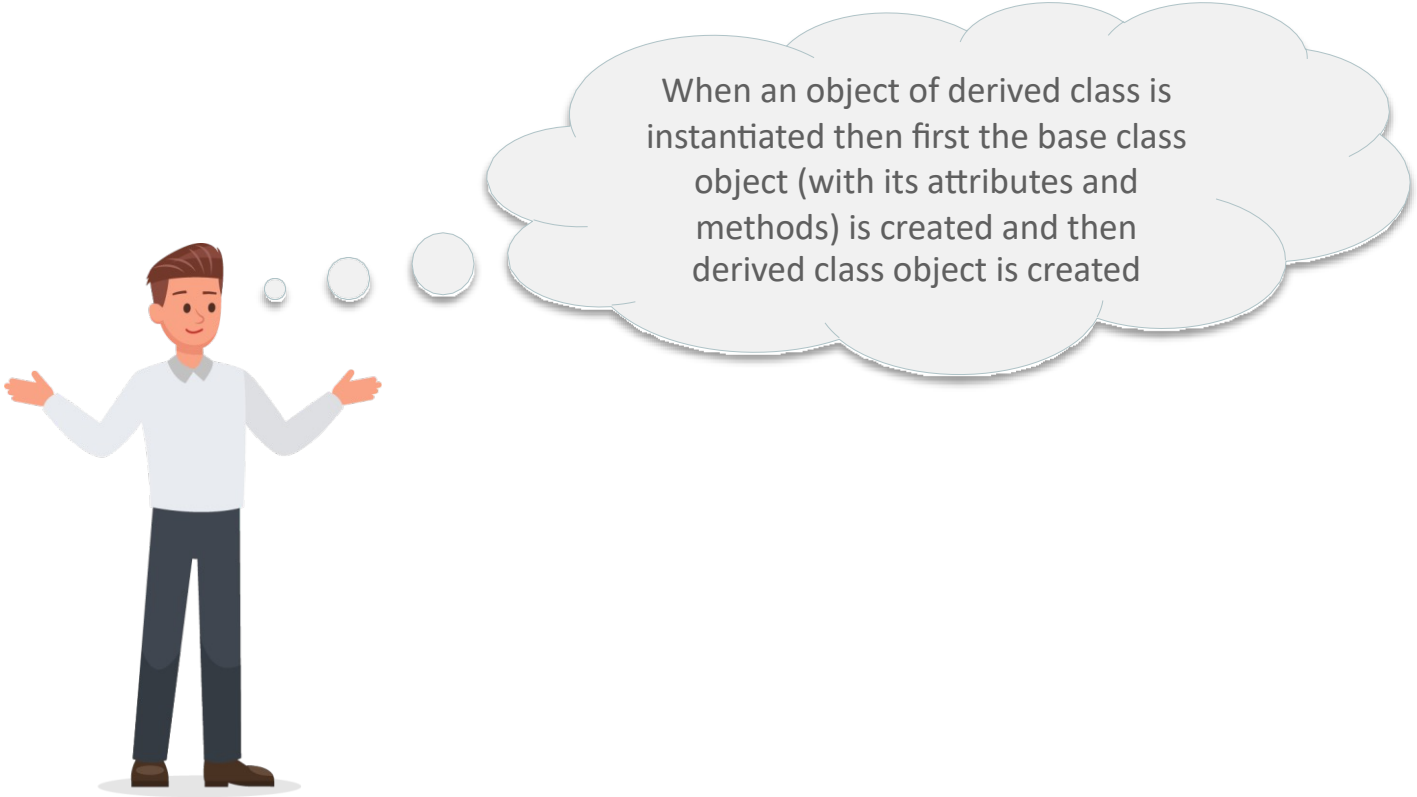
```
Class Animal {
        // Type : Not Human
        // Lives : On Land
        // Gives Birth :
    }
```

```
Class Aquatic extends Animal
{
        // Lives : In Water
        // Gives Birth :
}
```

Inheritance



Inheritance



Inheritance – Sample Program

```java
class baseClass {
    int x = 220;

    public void displayX() {
        System.out.println("Value of x : " + x);
    }
}

public class derivedClass extends baseClass {
    public static void main(String args[]) {
        derivedClass d1 = new derivedClass();
        d1.displayX();
    }
}
```

- In Java, “**extends**” keyword is used for extending the properties of base class to derived class.

Inheritance – Sample Program

```java
class baseClass {
    int x = 220;

    public void displayX() {
        System.out.println("Value of x : " + x);
    }
}

public class derivedClass extends baseClass {
    public static void main(String args[]) {
        derivedClass d1 = new derivedClass();
        d1.displayX();
    }
}
```

- In Java, “**extends**” keyword is used for extending the properties of base class to derived class.
- Base class defines an attribute X and has a method **displayX()**

Inheritance – Sample Program

```
class baseClass {
    int x = 220;

    public void displayX() {
        System.out.println("Value of x : " + x);
    }
}

public class derivedClass extends baseClass {
    public static void main(String args[]) {
        derivedClass d1 = new derivedClass();
        d1.displayX();
    }
}
```

- In Java, "**extends**" keyword is used for extending the properties of base class to derived class.
- Base class defines an attribute X and has a method **displayX()**
- Derived class is extending the base class and calling the **displayX()** to display the base class attribute

Multiple inheritance is not possible in Java. This means, one class can not be derived from multiple classes.

In Class Questions

1. Why do we need Inheritance? What are its advantages?

In Class Question - Solution

1. Why do we need Inheritance? What are its advantages?

Solution: Inheritance is used to reuse the common code present in class. For example, class “A” can make use of another class B's attributes and methods as if they are written class A. Modularity and code reusability are the advantages of Inheritance.

Constructors in Inheritance

baseClass Constructor

```
class baseClass {
    int x = 220;

    baseClass() {
        System.out.println("In the base class...");
    }

    public void displayX() {
        System.out.println("Value of x : " + x);
    }
}

public class derivedClass extends baseClass {
    derivedClass() {
        System.out.println("In the derived class...");
    }

    public static void main(String args[]) {
        derivedClass d1 = new derivedClass();
        d1.displayX();
    }
}
```

1

Constructors in Inheritance

baseClass Constructor

derivedClass Constructor

```
class baseClass {
    int x = 220;

    baseClass() {
        System.out.println("In the base class...");
    }

    public void displayX() {
        System.out.println("Value of x : " + x);
    }
}

public class derivedClass extends baseClass {
    derivedClass() {
        System.out.println("In the derived class...");
    }

    public static void main(String args[]) {
        derivedClass d1 = new derivedClass();
        d1.displayX();
    }
}
```

1

2

In Class Question

1. What will be the output of the following program? Which constructor will be called first? Is it from base class constructor or derived class constructor?

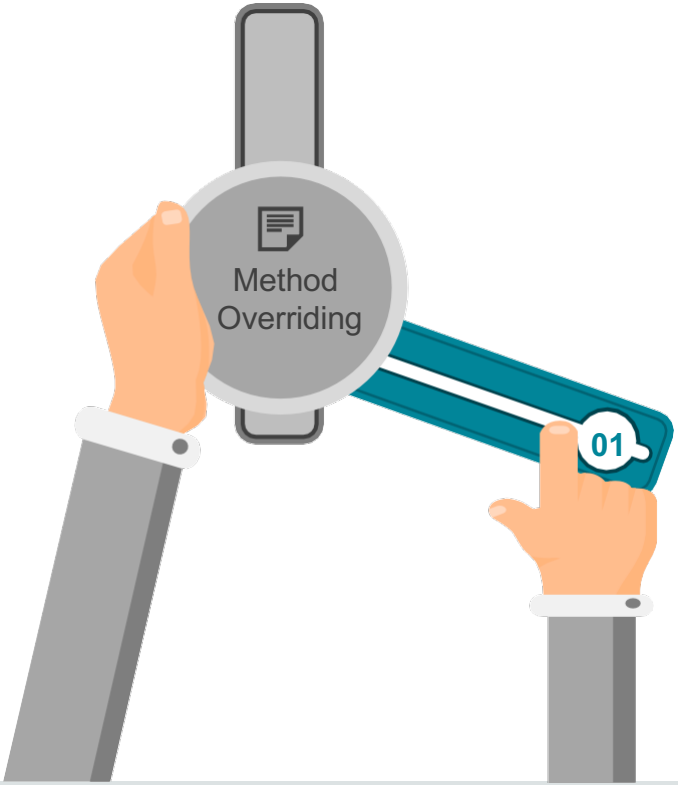
```
class baseClass {
    int x = 220;

    baseClass() {
        System.out.println("In the base class...");
    }

    public void displayX() {
        System.out.println("Value of x : " + x);
    }
}

public class derivedClass extends baseClass {
    derivedClass() {
        System.out.println("In the derived class...");
    }

    public static void main(String args[]) {
        derivedClass d1 = new derivedClass();
        d1.displayX();
    }
}
```

In Class Question - Solution

1. What will be the output of the following program? Which constructor will be called first? Is it from base class constructor or derived class constructor?

Solution: The output of the program: Value of x = 220. Base class constructor will be called first. It is from derived class constructor.

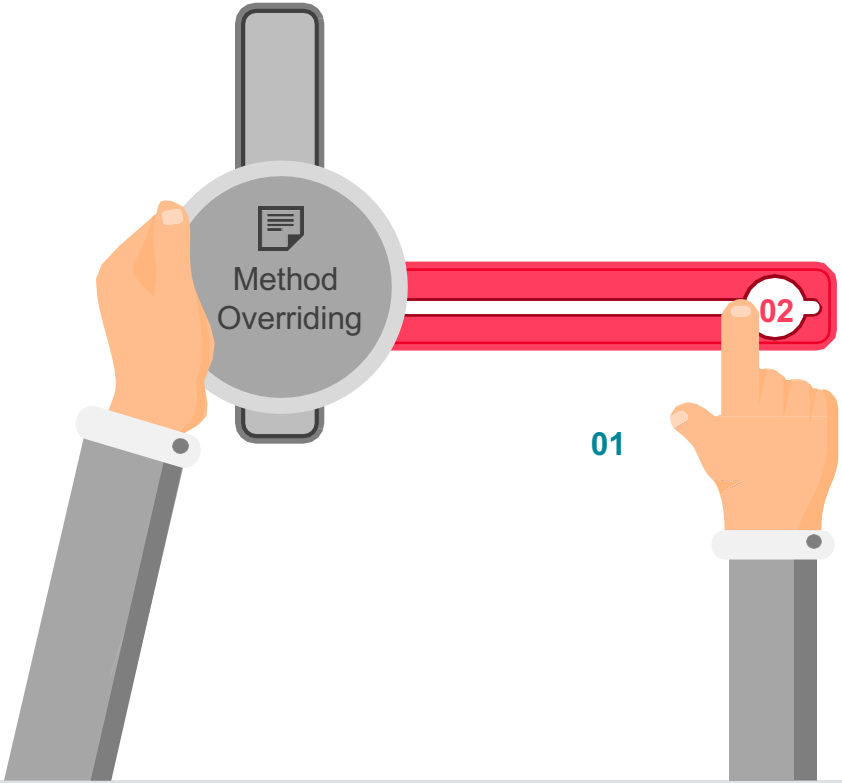
Method Overriding



Writing the base class method in the derived class is called as Method Overriding

01

Method Overriding



The reason to have this is to have the implementation of the derived class for the same method name

02

Writing the base class method in the derived class is called as Method Overriding

01

In Class Question

1. What is the difference between method overloading and method overridding? When are they required and why are they used?

In Class Question - Solution

1. What is the difference between method overloading and method overridding? When are they required and why are they used?

Solution: Many methods having the same method name with different arguments / parameters is called method overloading. Having the same method name in the base class and derived class is method overriding.

Runtime Polymorphism

```
class baseClass {
    int x = 220;

    baseClass() {
        System.out.println("In the base class...");
    }

    public void display() {
        System.out.println("In the base class display()");
    }
}

public class derivedClass extends baseClass {
    derivedClass() {
        System.out.println("In the derived class...");
    }

    public void display() {
        System.out.println("In the derived class display()");
    }

    public static void main(String args[]) {
        baseClass d1 = new derivedClass();
        d1.display();
    }
}
```

Overriding display()

Since Object is of derived class, the display method defined in child class will be called

In Class Question

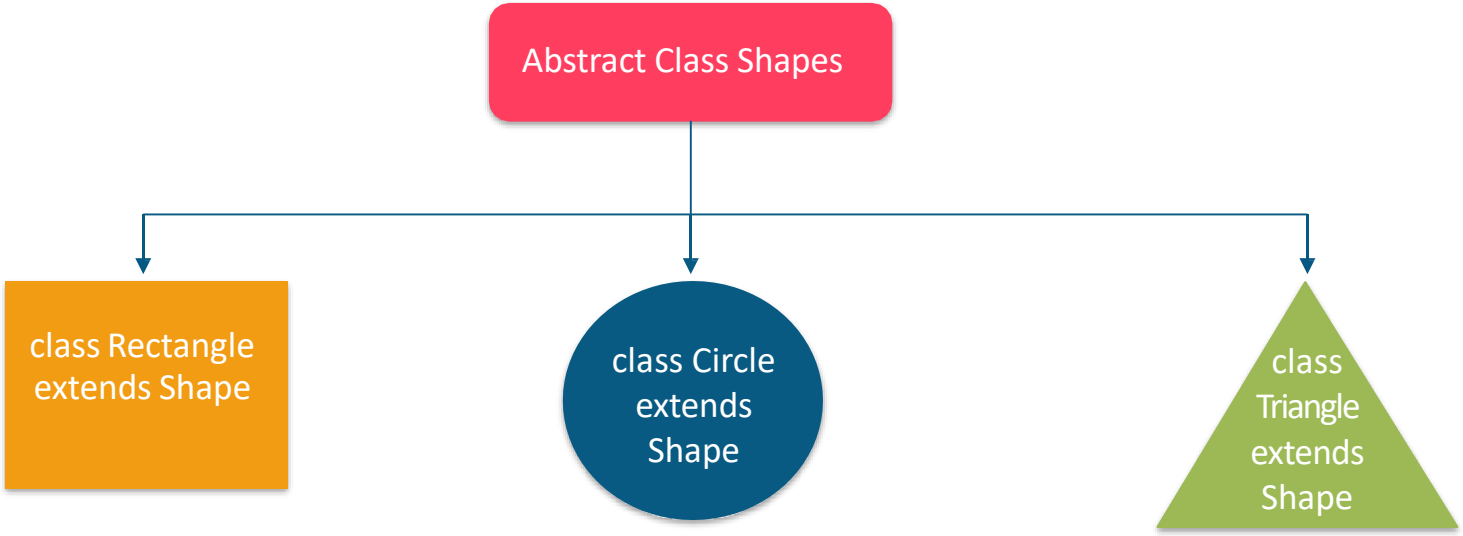
1. Can you come up with the output of the program listed above. Here base class variable is having the object of the derived class. There is display() function in both the classes. Which display method would be called?

In Class Question - Solution

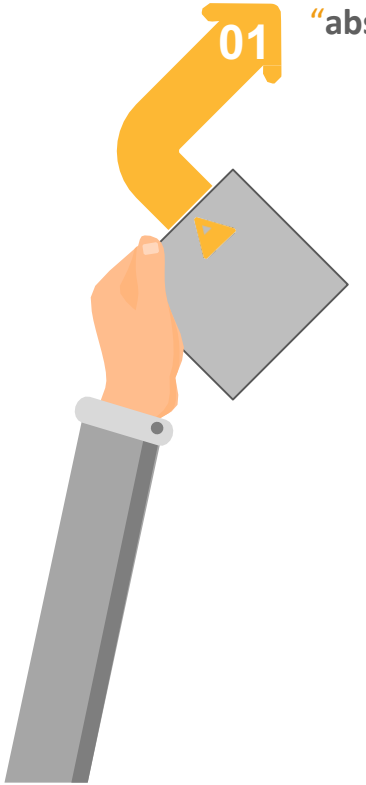
1. Can you come up with the output of the program listed above. Here base class variable is having the object of the derived class. There is display() function in both the classes. Which display method would be called?

Solution: Derived class display() method would be called as the object is of derived type. Many times, at run time you will get to know the object type and the corresponding method of the class would be called, hence runtime polymorphism.

Abstract Class

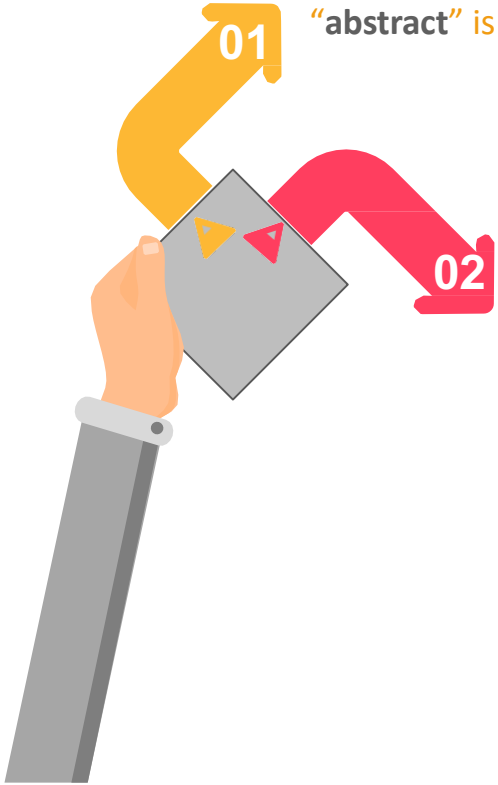


Abstract Class (contd.)



"**abstract**" is a keyword in Java.

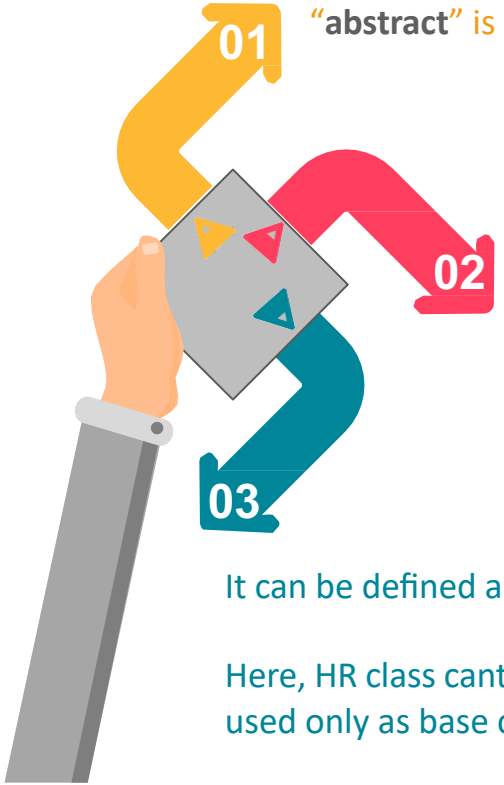
Abstract Class (contd.)



"**abstract**" is a keyword in Java.

During the design, if the requirement is just to have a class as a base class and should not be instantiated by the user then abstract class can be defined

Abstract Class (contd.)



"**abstract**" is a keyword in Java.

During the design, if the requirement is just to have a class as a base class and should not be instantiated by the user then abstract class can be defined

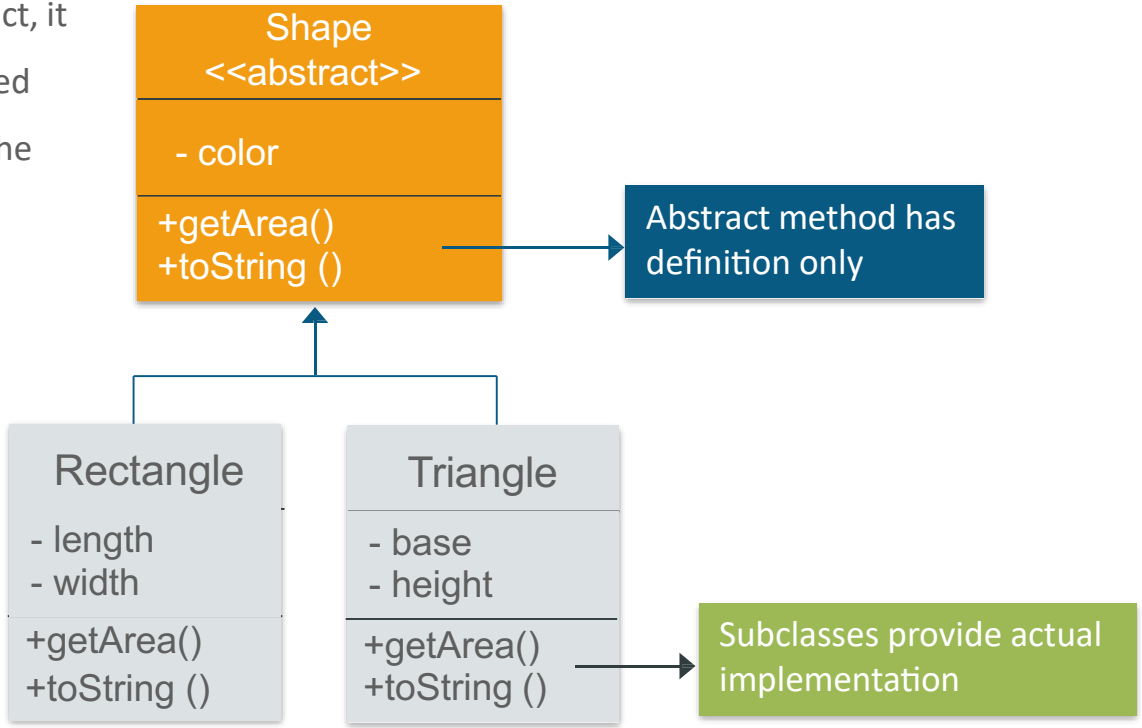
It can be defined as:

```
abstract class HR{}
```

Here, HR class cant be instantiated by the user. It can be used only as base class

Abstract Class and Methods

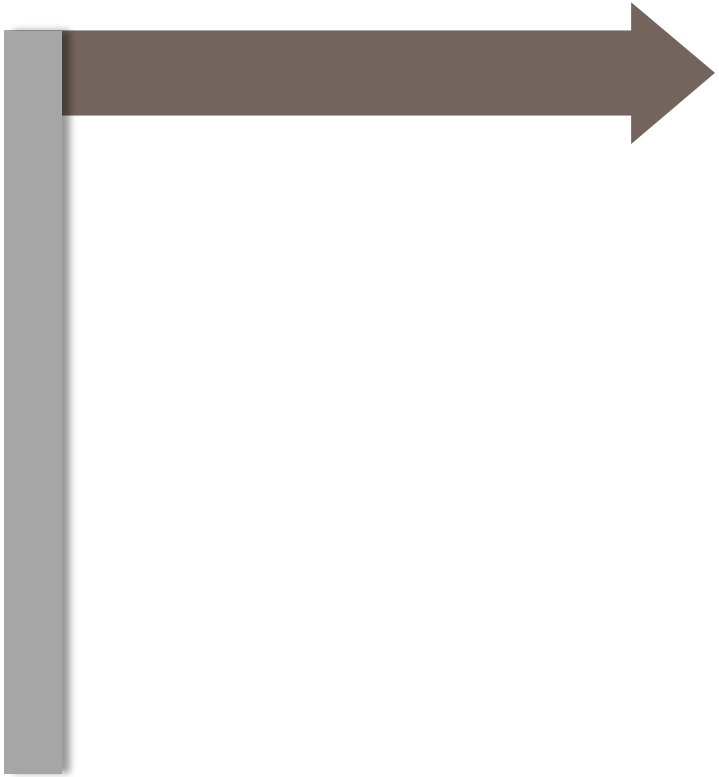
- When a method is defined as abstract, it will not have any body. A class derived from this class must implement all the abstract methods



Abstract Methods – Sample Program

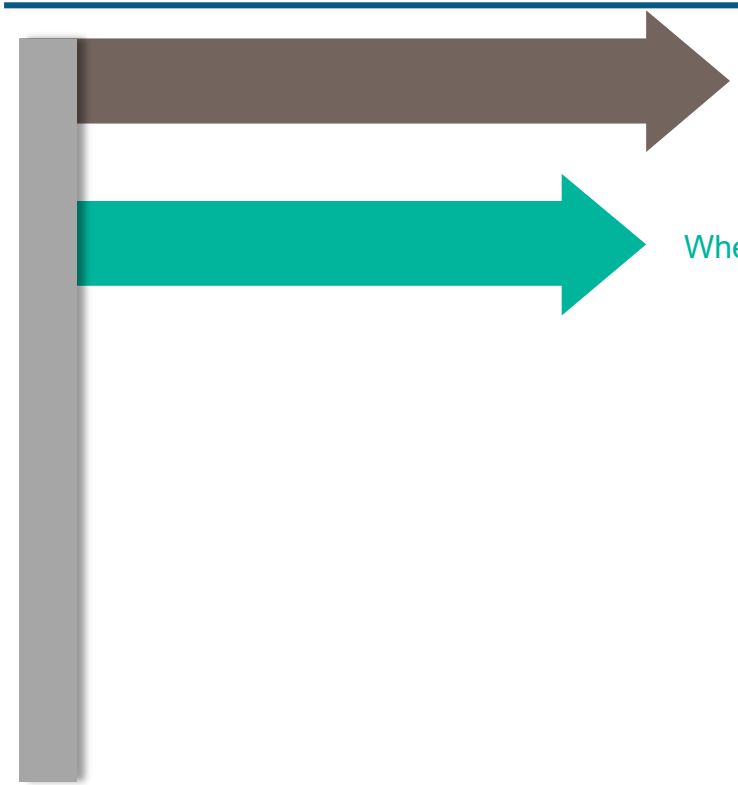
```
abstract class abstract1 {
    abstract void test();
}

public class abstract_demo extends abstract1 {
    void test() {
        System.out.println("in the test...");
    }

    public static void main(String arg[]) {
        abstract_demo a1 = new abstract_demo();
        a1.test();
    }
}
```

Final



"**final**" is a keyword in Java. **final** keyword can be used for attribute, method or for a class

Final



"**final**" is a keyword in Java. **final** keyword can be used for attribute, method or for a class

When **final** is used for **attribute**, it works as a **constant**.

Final

"**final**" is a keyword in Java. **final** keyword can be used for attribute, method or for a class

When **final** is used for **attribute**, it works as a **constant**.

It can be defined as:

```
final int x = 230;
```

Final

"**final**" is a keyword in Java. **final** keyword can be used for attribute, method or for a class

When **final** is used for **attribute**, it works as a **constant**.

It can be defined as:

```
final int x = 230;
```

In the code, **x** value is defined as **230** and it can't be changed in the entire program to some other integer. If done, compiler will throw an error

Final

"**final**" is a keyword in Java. **final** keyword can be used for attribute, method or for a class

When **final** is used for **attribute**, it works as a **constant**.

It can be defined as:

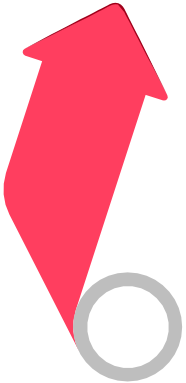
```
final int x = 230;
```

In the code, **x** value is defined as **230** and it can't be changed in the entire program to some other integer. If done, compiler will throw an error

When final is defined used for class then that class can't be extended. That means it can't be used for derived classes

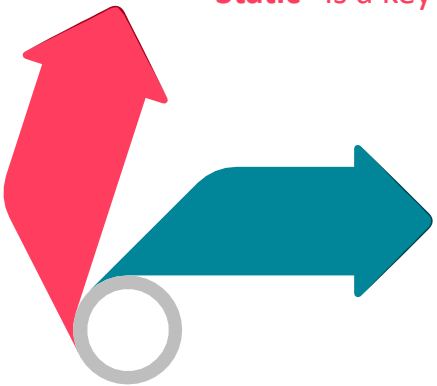
```
final class Employee
```

Static



"**Static**" is a keyword in Java. It can be used for an attribute or a method or a block

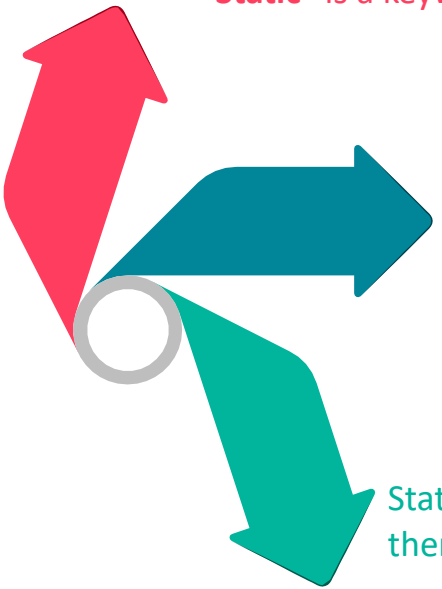
Static



"**Static**" is a keyword in Java. It can be used for an attribute or a method or a block

Static attribute / method / block gets loaded when the class gets loaded. This is before the object gets created

Static



"**Static**" is a keyword in Java. It can be used for an attribute or a method or a block

Static attribute / method / block gets loaded when the class gets loaded. This is before the object gets created

Static attribute / method / block is from class not from the object. Hence to access them object creation is not necessary

Program on Static

```
public class static_demo {
    static int x = 230;

    public static void DisplayMessage() {
        System.out.println("In the static function...");
    }

    static {
        System.out.println("In the static block...");
    }

    public static void main(String args[]) {
        System.out.println("Value of static variable : " + static_demo.x);
        static_demo.DisplayMessage();
    }
}
```

Did You know?

Following things can be marked as Static:
→ Methods
→ Variables
→ Nested Classes
→ Initialization Block

Following things cannot be marked as Static:
→ Constructors
→ Classes
→ Interface
→ Method local INN.class
→ Inner Class methods and Variables
→ Local Variable