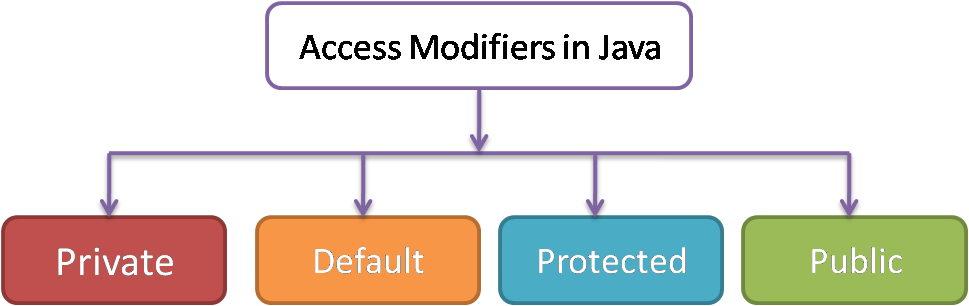
# Unit 3. Classes and Objects

| 3.1. Simple Class, Field  3.2. Access Controls, Object creation  3.3. Construction and Initialization  3.4. Methods, this pointer  3.5. Overloading Methods & Constructors.  3.6. Static members, static block, static class  3.7. Inheritance, super, abstract class, overriding methods  3.8. Interfaces  3.8.1. Introduction to Interfaces.  3.8.2. Interface Declaration.  3.8.3. Inheriting and Hiding Concepts.  3.8.4. Inheriting, Overloading and Overriding Methods.  3.8.5. Interfaces Implementations. |
| --- |

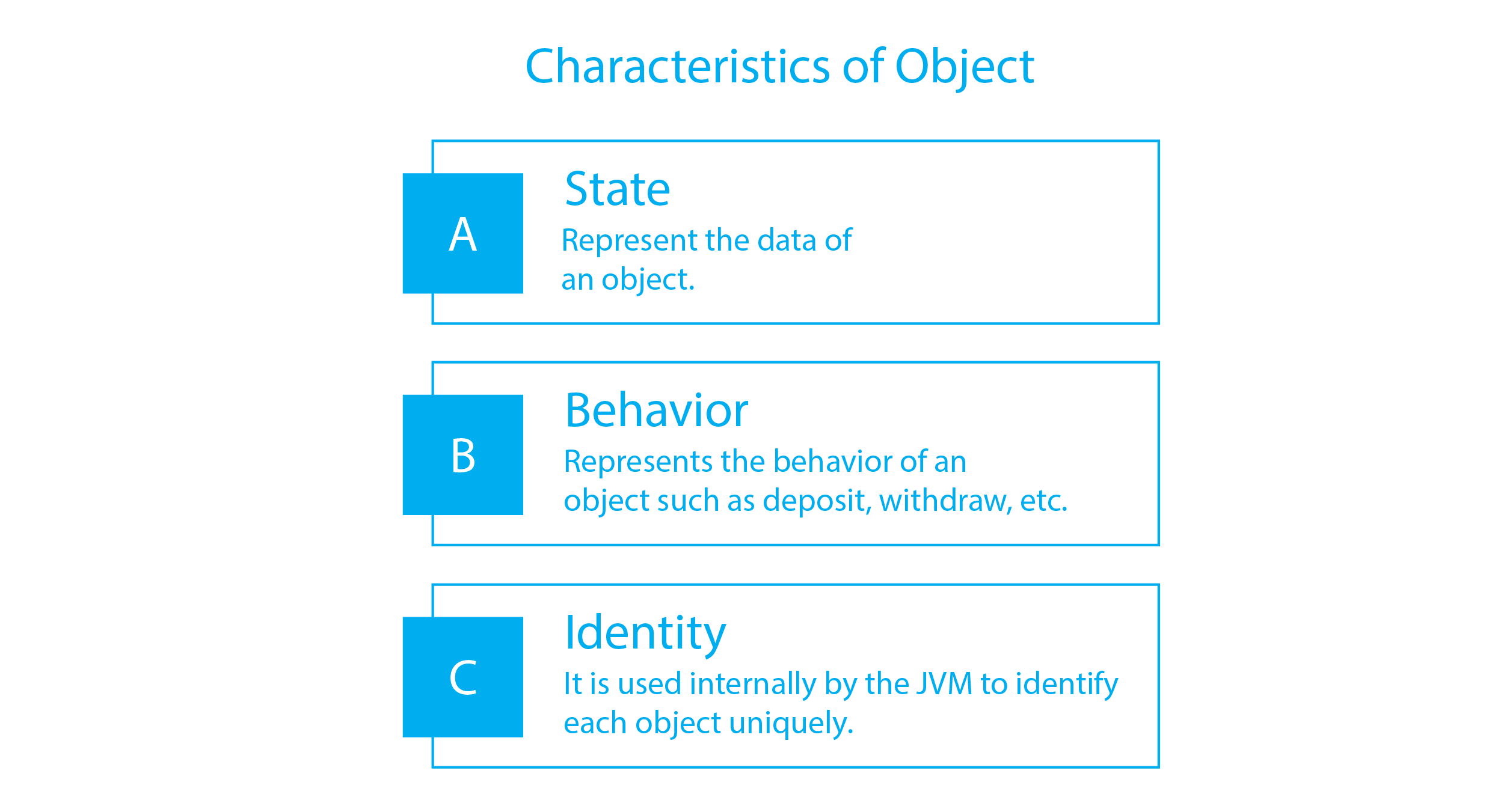
**3.1. Simple Class, Field**

**3.2. Access Controls, Object creation**

* **Access controls / Access modifiers / Access specifiers / visibility modes :**
* Access modifiers define the scope or visibility of the members of a class, these members can be variables, constructor or method.
* Access modifiers determine whether other classes can use a particular field or invoke a particular method can be public,private,protected,default.



* **Private :**
* Private keyword is used to specify the private access modifiers.
* The methods or data members declared as private are accessible only within its own class.
* Top level classes or interfaces can not be declared as private.
* **Default :**
* When no Access modifier is specified for a class,method or data member- it is said to be having the default access modifier by default.
* **Protected :**
* Protected keyword is used to specify the protected access modifier.
* The protected access modifier is accessible within package and outside the package but through inheritance only.
* **Public :**
* Public keyword is used to specify the public access modifier.
* The public access modifier is accessible everywhere.
* It has the widest scope among all other modifiers.
* **Object creation :**
* Object is an instance of java class.
* An entity that has state and behavior is known as an object.
* **An object has three characteristics :**



* **Syntax to declare an object :**

<Class\_name> <varaible\_name> = new <class\_constructor>

* **Example :**

| class student  {  **// Member function**  **void detail( )**  **{**  **System.out.println("hello students !");**  **}**  **// Main function**  public static void main(String args[ ])  {  **student s1=new student( ); // Object declaration**  s1.detail( );  }  } |
| --- |

* There are two ways to initialize an object.

1. By reference variable
2. By method

**3.3. Construction and Initialization**

**3.4. Methods, this pointer**

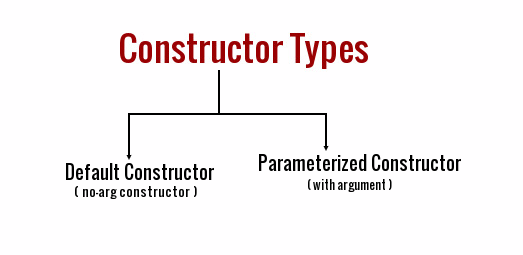
* **Java methods :**
* It is a collection of statements that are grouped together to perform an operation.
* It takes some parameters, performs some computatios and the optionally return a value or an object.
* It is used to perform certain actions.

**3.5. Overloading Methods & Constructors.**

* **Polymorphism :**
* Polymorphism is derived from greek word i.e. poly means many and morphism means form.
* Polymorphism is a concept of OOP in which object is treated in different ways.
* In other words,Polymorphism means more than one function with same name, with different working.
* In java, we use **function overloading** and **function overriding** to achieve polymorphism.
* **Overloading Methods / function overloading :**
* Function Overloading is the process of having two or more function with the same name,but different in parameters is known as function overloading.

| **Example** | **Output** |
| --- | --- |
| class test  {  **void add(int a,int b)**  {  System.out.println("first :: "+(a+b));  }  **void add(int a,int b,int c)**  {  System.out.println("second :: "+(a+b+c));  }  public static void main(String args[ ])  {  test t1=new test( );  **t1.add(3,2);**  **t1.add(3,2,1);**  }  } | first :: 5  second :: 6 |

* **Constructors :**
* Constructor is a special type of function which is used to initialize an object.
* Constructor Is a member function which has the same name of class name
* Constructor must have no explicit return type.



1. **Default constructor :**

* A default constructor is a constructor that either has no parameters or if it has parameters,all the parameters have default value.
* If there is no constructor in a class, compiler creates a default constructor automatically.
* In simple word , a constructor which does not have any parameters is called default constructor.
* **Syntax :**

| class class\_name  {  class\_name( )  {  // Statements  }  } |
| --- |

| **Example** | **Output** |
| --- | --- |
| class test  {  **//Default constructor**  **test( )**  **{**  **System.out.print("Hello Friends ! ");**  **}**    public static void main(String args[ ])  {  test t1=new test( );  }  } | Hello Friends ! |

1. **Parameterized constructor :**

* Parameterized constructor in java are the constructor in a class which have one or more than one argument.
* There can be more than one Parameterized constructor in a class.
* **Syntax :**

| class class\_name  {  class\_name(parameters)  {  // Statements  }  } |
| --- |

| **Example** | **Output** |
| --- | --- |
| class test  {  // **Parameterized constructor 1**  **test(int a,int b)**  **{**  **System.out.println("a :: "+a);**  **System.out.println("b :: "+b);**  **}**  **// Parameterized constructor 2**  **test(int no,String name)**  **{**  **System.out.print(no+" "+name);**  **}**    public static void main(String args[ ])  {  test t1=new test(10,20);  test t2=new test(121,"ram");  }  } | a :: 10  b :: 20  121 ram |

**3.6. Static members, static block, static class**

* **Static members :**
* static members are those which belongs to the class and you can access the members without creating an object.
* It can access static data member and can change the value of it.
* **Syntax :**

| Access\_modifier static return\_type function\_name  {  // Body of function  } |
| --- |

| **Example** | **Output** |
| --- | --- |
| class test  {  static String a="motu";  **// Static member**  **static void update( )**  **{**  **a="patlu";**  **}**  void display( )  {  System.out.println("old value : motu");  System.out.println("new value : "+a);  }  public static void main(String args[ ])  {  test t1=new test( );  update( );  t1.display( );  }  } | old value : motu  new value : patlu |

* **static block :**
* It is used to initialize a static data members.
* It is executed before main( ) function at the time of class loading.
* **Syntax :**

| static  {  //Statements  } |
| --- |

| **Example** | **Output** |
| --- | --- |
| class test  {  static  {  System.out.println("101");  System.out.println("ram");  System.out.println("90000");  }    public static void main(String args[ ])  {  // none  }  } | 101  ram  90000 |

* **static class :**
* Static
* **Syntax :**

| class outer\_class  {  static class inner\_class  {  // data member  // member function  }  } |
| --- |

| **Example** | **Output** |
| --- | --- |
| class outer  {  static class inner  {  void data( )  {  System.out.println("Static class");  }  }    public static void main(String args[ ])  {  outer.inner obj=new outer.inner( );  obj.data( );  }  } | Static class |

**3.7. Inheritance, super, abstract class, overriding methods**

* **Inheritance :**
* Inheritance is one of the key feature of OOP that allows us to create a new class from existing class.
* The new class which is created by existing class is known as **sub class**,**child class** or **derived class**.
* the existing class from where child class is derived is known as **super class**,**parent class** or **base class**.
* The **extends** keyword is used to perform inheritance in java.
* **Syntax :**

| class Base  {  // Fields and Method  }  class Derived **extends** Base  {  // Fields and Method of Derived class  // Fields and Method of Base class  } |
| --- |

* **Types of inheritance :**
* There can be three types of inheritance in java :

1. Single inheritance
2. Multilevel inheritance
3. Hierarchical inheritance

* In java programming, multiple and hybrid inheritance is supported by interface only.

| **Single inheritance** |  |
| --- | --- |
| **Syntax** |  |
| **Example**  **Output ::**   | **enter a : 55**  **enter b : 45**  **sum is :: 100** | | --- | | import java.util.Scanner;  class one  {  int a,b;  void input( )  {  Scanner s=new Scanner(System.in);  System.out.print("enter a : ");  a=s.nextInt( );  System.out.print("enter b : ");  b=s.nextInt( );  }  }  class two extends one  {  int c;  void output( )  {  c=a+b;  System.out.println("sum is :: "+c);  }  }  class main  {  public static void main(String args[ ])  {  two t=new two( );  t.input( );  t.output( );  }  } |

| **Multilevel inheritance** |  |
| --- | --- |
| **Syntax** |  |
| **Example**  **Base Class**  **Intermediate class**  **derived class** | class one  {  void input1( )  {  System.out.println("Base Class");  }  }  class two extends one  {  void input2( )  {  System.out.println("Intermediate class");  }  }  class three extends two  {  void input3( )  {  System.out.println("derived class");  }  }  class main  {  public static void main(String args[ ])  {  three t=new three( );  t.input1( );  t.input2( );  t.input3( );  }  } |

| **Hierarchical inheritance** |  |
| --- | --- |
| **Syntax** |  |
| **Example**  **base class**  **derived class 1**  **base class**  **derived class 2** | class one  {  void base( )  {  System.out.println("base class");  }  }  class two extends one  {  void derived1( )  {  System.out.println("derived class 1");  }  }  class three extends one  {  void derived2( )  {  System.out.println("derived class 2");  }  }  class main  {  public static void main(String args[ ])  {  two t1=new two( );  three t2=new three( );  t1.base( );  t1.derived1( );  t2.base( );  t2.derived2( );  }  } |

**3.8. Interfaces**

**3.8.1. Introduction to Interfaces.**

* Interface is a collection of Abstract methods and data members which are static and final.
* There can be only abstract methods in java interface that means only declaration not definition.
* It is used to achieve fully abstraction and multiple inheritance in java.
* The java compiler adds public and abstract keywords before the interface method and public,static and final keyword before data members.

**3.8.2. Interface Declaration.**

**3.8.3. Inheriting and Hiding Concepts.**

**3.8.4. Inheriting, Overloading and Overriding Methods.**

**3.8.5. Interfaces Implementations.**