**Accessors and Mutators**

Acessor /get method

* It will get the information about the object without changing anything
* They allow you to peek at the state of an object without disturbing it, which is crucial for keeping your code safe and organized.
* They have the return type
* Accessor method do not modify the state of object
* Accessor and mutator method should be declared as public to access other classes.

Ex public string getName()

{

return name;

}

Mutator/ set

-Mutator method used to modify the value of objects attribute.

-mutator do not return any value so it is void.

Ex

Public void setName(String name)

{

This.name=name;

}

**Static Keyword**

* Static Variable
* The variable declared with static keyword are called static variable and class variable.
* The are allocated memory once ,at the start of the program execution , and remain in existence until the program terminates.
* Static variables can be accessed by class name only.
* Static Method
* The method declared with static keyword is static method.
* They can be called directly using class name.
* Cannot Access Non-Static Members : Static methods cannot access instance variable or instance methods directly they can only access other static members.
* Static Block
* A static block is a block of code enclosed in ‘braces’ {} and preceded by the static keyword.
* It is executed when class is loaded into the memory .
* Often used for static variable initialization or other one-time initialization that need to be performed when the class is loaded.

**Variables**

* Variables
* Local Variables:
* Variables declared inside the method, constructor, Block
* Scope is limited to method or Block
* Lifetime exits only during the execution of the block or method
* Instance Variable
* Variables declared within class but outside any method, constructor, block
* Each instance of the class (object) has its own copy of instance variable.
* Lifetime Exits as long as Object exits
* They are initialized when object is created and destroy when it is garbage collected.
* Int max;
* Static Variable (Class Variable)
* Variable declared with ‘static ‘ keyword within class but outside a method, constructor or block.
* Scope Shared among all instance of class (Object) there is only one copy per class.
* Lifetime exits as long as class remain loaded in memory ;They are initialized when the class is loaded and destroyed when the class is unloaded
* Eg static int max;
* Final /Constant Variable
* Variable declared with ‘Final ‘ keyword whose value cannot be changed once initialized
* Scope is similar to the local,static,instance depends on where they declared
* Eg final int max=10;
* toString Mehod
* This method returns the String representation of Object
* It’s method of Object class
* Eg public String toString()
* {

return “Name”+name+””+”Address”+address;

* }
* This Keyword
* ‘this’ keyword is a reference variable that refers to the current instance of the class .
* It used to refer to instance variable , invoke methods , pass the current objects as parameter to other methods.
* Accessing Instance variable
* ‘this’ to refer to instance variable of the current object.
* This is particularly useful when there is a local variable with the same name as an instance variable, to distinguish between them.

-eg

Public class demo

{

Private int x;

Public demo(int x)

{

This.x=x;

}

}

* Invoking Methods
* You can use ‘this’ to invoke other methods of current object.
* This is often used to invoking the overloaded constructor or other methods of the same class.
* Eg

Public Class Myclass

{

Private int x;

Public Myclass()

{

This(0);

}

Public Myclass(int x)

{

This.x=x;

}

}

Calling another constructor or other method within the same class.

* Returning the current object
* Methods can return the current object using this keyword.
* Where multiple methods can be called on single object in a single statement. Aka fluent interface or method chaining.
* Eg

**package** ThisKeyword;

**public** **class** MethodChaning {

**private** String name;

**private** **int** age;

**public** MethodChaning setName(String name)

{

**this**.name=name;

**return** **this**;

}

**public** MethodChaning setAge(**int** age)

{

**this**.age=age;

**return** **this**;

}

**public** **void** printDetails()

{

System.***out***.println("Name"+name+""+"Age"+age);

}

**public** **static** **void** main(String[] args)

{

MethodChaning m=**new** MethodChaning();

m.setName("AA").setAge(12).printDetails();

}

}

* Passing the current Object as Parameter

-You can pass the current object as a parameter to other methods or constructor.

public class MyClass {

private int x;

public MyClass(int x) {

// Some initialization

doSomething(this); // Pass the current object to another method

}

private void doSomething(MyClass obj) {

// Perform operations using the passed object

}

}

**Association**

* In Java association is a Relationship between two classes that describes how objects of one class are connected or related to objects of another class.
* One To One
* One To Many
* Many To One
* Many To Many

Note: There can be two types of relationship in OOPS

[ is a ] Inheritance

[ has a ] Containment

Containment : One Object contains another object as a part of it’s attribute is known as containment it can be of two forms.

i] Aggregation [loosely-coupled]

ii] compostion [Tightly-coupled]

i] Aggregation : It is loosely-coupled relation one can exists other independently it is weak association.

Ii] Composition : Tightly –coupled relation they can not exists independent of each other it is a strong association.

Loosely-coupled/weak Tightly-coupled/strong

car car

Container Object Container Object

car

Contained Object

Contained Obj

There is Contained object is There is Contained object is

Independent of the container obj dependent of the container obj

**Access Modifiers**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Access Modifiers | Within  class | Within  Package | Outside package by subclass only | Outside  Package |
| Private | Y | N | N | N |
| Default | Y | Y | N | N |
| Protected | Y | Y | Y | N |
| Public | Y | Y | Y | Y |

**Inheritance**

**Defination**: Inheritance is mechanism I java in which subclass inherits the fields and methods of superclass and can also have its own fields and methods.

* In java Inheritance can be achieved by using the **‘extends’** keyword.
* Types of inheritance
* Single level
* Multi level
* Hierarchical inheritance
* Access Modifiers and Inheritance
* Public and Protected members of the superclass are accessible by the subclass.
* Private members of the superclass are not accessible by the subclass.
* Default (package-private) members of the superclass are accessible by the subclass only if they are in the same package.
* Constructor Inheritance
  + Constructors are not inherited by subclasses ,but the subclass constructor implicitly calls the superclass constructor.
  + If no constructor is explicitly defined in the subclass , the compiler insert a defult constructor that calls the sperclass’s default constructor SUPER().
* Method Overriding
* Subclasses can provide specific implementation for a method that is already defined in the superclass. AKA method Overriding.
* Subclass method must have the same signature (name and parameter) as the superclass method.
* Method overriding can be achieved by using @annotation (it’s optional)
* The return type of the overriding method must be the same as ,or subtype of the return type of the overridden method.
* the method in the subclass cannot throw a checked exception that is broader than the one thrown by the superclass method.
* Super keyword
* The Super keyword is used to refer to the superclass members (fields,.methods,constructor )from within the subclass.
* It is also used to invoke the superclass constructor explicitly.
* ‘super()’ calls the superclass constructor ,while ‘super.method()’ calls the superclass method.
* Final keyword and Inheritance
* The ‘ Final ‘ keyword can be applied to classes, methods, variables.
* A Final class cannot be subclassed.
* A Final method cannot be overridden by subclasses.
* A Final variable cannot be reassigned a new value once initialized.
* Object Class and inheritance
  + Every class in java is a subclass of the **‘Object’** class either directly or indirectly.
  + The Object class is the root of the class hierarchy in java and provides several methods that are inherited by all classes such as **‘ toString’, equals( ) ,** **hashCode().**

**Polymorphism**

* An ability of an object to take on different forms or behaviours depending on its context.
* In java polymorphism allows objects of different classes to be treated as objects of a common superclass through inheritance.
* Compile –time polymorphism / Static Binding/Early Binding
* Also known as method overloading
* Occurs when the method to be invoked is determined at compile time based on the method signature.
* Multiple methods with same name but different parameters can exist in the same class.
* Run-time Polymorphism/ Dynamic Binding/ Late Binding
* Also known as method overriding
* Occurs when the method to be invoked is determined at runtime based on the actual type of the object.
* Involves a superclass reference variable pointing to a subclass object.
* The overridden method in the subclass provides a specific implementation of the method defined in the superclass.
* Method Overloading
* Method overloading allows a class to have multiple methods with same name but different parameter list/
* The compiler determines which overloaded method to call based on the number and types of argument provided.
* Overloaded method must have unique signitures , which includes the method name and parameter types.
* Method Overriding
* Method overriding allows a subclass to provide a specific implementation of a method defined in its superclass.
* The subclass method must have the same name ,return type ,and parameter list as the overridden method in the superclass.
* Annotations like @Override can be used to indicate that a method is intended to override a superclass method
* Dynamic Method Dispatch
* Dynamic method dispatch is the mechanism by which the correct version of an overridden method is called at runtime.
* It allows for method invocation based on the type of the object rather than the reference variables type.
* Dynamic method dispatch is achieved through inheritance and method overridind.
* Casting
* Casting allows you to treat an object as an instance of another class or interface.
* Upcasting is implicit and is done automatically by the compiler when assigning a subclass object to a superclass reference variable.
* superclass reference variable to a subclass reference variable.
  + UpCasting :
* is the typecasting of child object to parent object it can be done implicitely .it gives the flexibility to access the parent class members but it is not possible to access all the child class members using this feature .we can access the overridden method only.

Parent

DownCasting

Explicitely

UpCasting

Implicitly

* + DownCasting:
* The Typecasting of parent object to child object
* It must be perform explicitely
* We forcefully cast parent to a child object.

Animal animal =new Dog();

Reference Variable Actual Object

Dog dog =(Dog) animal // downCasting

Here we accessing the methods of the child class using the parent class reference

For this we have to downcast the animal reference to the child dog object.

* Instanceof
* The ‘instanceof ‘ operator checks whether an object is an instance of particular class or interface.
* It **returns**  **True** if the object is instance of particular class if not **returns False**
* Its used to avoid **ClassCastException.**
* Covariant
* Java allows you to change return type in the “ Overriden Method “ as long as as new return type its “sub type “ of declared type

Eg: class Animal {

Animal giveBirth() {

System.out.println("Animal is giving birth");

return new Animal(); } }

class Dog extends Animal {

@Override

Dog giveBirth() { // Covariant return type

System.out.println("Dog is giving birth");

return new Dog();

} }

public class Main {

public static void main(String[] args) {

Animal animal = new Dog();

animal.giveBirth(); // Calls Dog's giveBirth() method } }

**Abstract Class**

Is the process of identifying the key aspects of an entity and ignoring the rest.

* It is declared with “ abstract ” keyword .
* It may have abstract and concrete method
* A method with abstract keyword is known as abstract method.
* Abstract method must be implemented by it’s child class .if the child class is unable to provide implementation then that class must be also “abstract”
* Instance of abstract class cannot be created.
* Abstract class can have “constructors “ and fields like any other class.
* A abstract class can be without any abstract method.
* There can be “final or static method “ abstract class never be “final”.

Eg: abstract class AbstractClass {

// Abstract method (no body)

abstract void abstractMethod();

// Concrete method with a body

void concreteMethod() {

// Method implementation

}

}

**Interface**

Is used to provide common design across classes that are not hierarchy it used to create loosely-coupled applications.

* Interface keyword used to create interface.
* All methods inside interface are abstract methods.
* There is no need to write abstract keyword.
* A class has to use “implements” keyword to use an interface.
* It is mandatory for those classes which implements interface .to write the implementation of abstract methods.