**BCA-DS-212: PROGRAMMING IN JAVA *BCA IV Semester***

**UNIT 1**

**1. Imperative programming paradigm:** It is one of the oldest programming paradigm. It features close relation to machine architecture. It is based on Von Neumann architecture. It works by changing the program state through assignment statements. It performs step by step task by changing state. The main focus is on how to achieve the goal. The paradigm consist of several statements and after execution of all the result is stored.

**Advantages:**

1. Very simple to implement
2. It contains loops, variables etc.

**Disadvantage:**

1. Complex problem cannot be solved
2. Less efficient and less productive
3. Parallel programming is not possible

**Types of Imperative Programming Paradigm**

1. **Procedural programming paradigm –**   
   This paradigm emphasizes on procedure in terms of under lying machine model. There is no difference in between procedural and imperative approach. It has the ability to reuse the code and it was boon at that time when it was in use because of its reusability.

Examples of **Procedural** programming paradigm:  
  
**C** : developed by Dennis Ritchie and Ken Thompson  
**C++** : developed by Bjarne Stroustrup  
**Java** : developed by James Gosling at Sun Microsystems  
**ColdFusion** : developed by J J Allaire  
**Pascal** : developed by Niklaus Wirth

1. **Object oriented programming –**   
   The program is written as a collection of classes and object which are meant for communication. The smallest and basic entity is object and all kind of computation is performed on the objects only. More emphasis is on data rather procedure. It can handle almost all kind of real life problems which are today in scenario.

**Advantages:**

* Data security
* Inheritance
* Code reusability
* Flexible and abstraction is also present

Examples of **Object Oriented** programming paradigm:  
  
**Simula** : first OOP language  
**Java** : developed by James Gosling at Sun Microsystems  
**C++** : developed by Bjarne Stroustrup  
**Objective-C** : designed by Brad Cox  
**Visual Basic .NET** : developed by Microsoft  
**Python** : developed by Guido van Rossum  
**Ruby** : developed by Yukihiro Matsumoto   
**Smalltalk** : developed by Alan Kay, Dan Ingalls, Adele Goldberg

**2. Declarative programming paradigm:**   
It is divided as Logic, Functional, Database. In computer science the *declarative programming* is a style of building programs that expresses logic of computation without talking about its control flow. It often considers programs as theories of some logic.It may simplify writing parallel programs. The focus is on what needs to be done rather how it should be done basically emphasize on what code is actually doing. It just declares the result we want rather how it has be produced. This is the only difference between imperative (how to do) and declarative (what to do) programming paradigms. Getting into deeper we would see logic, functional and database.

* **Logic programming paradigms –**   
  It can be termed as abstract model of computation. It would solve logical problems like puzzles, series etc. In logic programming we have a knowledge base which we know before and along with the question and knowledge base which is given to machine, it produces result. In normal programming languages, such concept of knowledge base is not available but while using the concept of artificial intelligence, machine learning we have some models like Perception model which is using the same mechanism.   
  In logical programming the main emphasize is on knowledge base and the problem. The execution of the program is very much like proof of mathematical statement, e.g., Prolog

**Evolution of programming languages**

* Object oriented languages did not come on their own but were inspired by the need felt while using imperative languages.
* Non-OOP languages are mother of almost all programing languages. ALGOL(1960) is mother of almost all imperative languages like C while  LISP and SIMULA(1967) (used for simulation) are for object oriented programing languages like SmallTalk.
* SmallTalk(1969) is pure OOPL. It is not used for building object oriented systems but for research and development of object oriented languages.
* Then came imperative languages like Modula, Pascal and C while functional languages like Haskell (pure) and ML(impure)
* Clu(1970) and Ada are based on Abstract data type
* Then came C with classes(1980) that evolved to first widely used object oriented language C++. Cfont preprocessor was used to convert the C++ code to C code. Thus compiled the C++ code using C compiler and Cfont. This is called as bootstrapping.

**Evolution of OOP**

* Cfont converts C++ program to C program which is compiled using C compiler. It behaves like a preprocessor and not like compiler or translator.
* What is the difference between preprocessor and compiler ?
* Preprocessor : It converts one high level language to another high level language. e.g. C preprocessor, it processes anything starting with preprocessor directive ‘#’ like macros defined using #define.
* It does only text substitution
* May lead to errors
* Compiler /Translator : It converts a high level language to a low level language that is more closer to the machine.
* It requires language with proper grammar
* It has syntax analysis, semantic analysis, syntax directed translation, intermediate code generation, optimisation and target code generation.
* Other languages that evolved were:
* CLOS - (Object oriented version of LISP , not pure OOPL)
* Cecil - (developed in washington university, it is pure OOPL)
* Python - (procedural, object oriented, functional,scripting language)
* Java - (object oriented, imperative,scripting language)
* It has syntax similar to C and C++
* Along with language came the Virtual Execution Environment(VEE) because of which java lead to programming paradigm shift.
* x10 - It is upcoming language developed by IBM in 2010.
* It is comprehensive high-end object oriented programming language.
* It is used for concurrent programming. SInce systems today are multi-core, we require support for concurrency, high performance, high productivity computing on high end computers.
* aim of good programming language is to develop a system that is:
* Reusable
* Easy to program
* Safer and trustworthy
* PERCS (Productivity Easy to use Reliable Computer System)
* Object oriented programming has following features.
* Abstract Data Type (ADT)
* Inheritence
* Dynamic binding

Issues to be addressed

* Issue 1: OOPL: Pure and Mixed
* Our aim is not to have a pure object oriented programming but have a good system (reusable, easy to program, safer and trusted)
* Impure does not mean it is bad but it means it has something more that adds to the power.
* Pure languages were used for research and thats how many object oriented features were added to them. Impure languages are used for day to day development of good software systems.
* Issue 2: Major ingredients of OOPL
* Inheritance : It is fundamental to object oriented programming.
* what are we inheriting ? (will be answered in future classes)
* Dynamic binding : Dynamic means during runtime while static means during compile time. When binding is done at the linking phase it is called dynamic binding.

# Java OOPs Concepts

Object-Oriented Programming is a paradigm that provides many concepts, such as **inheritance**, **data binding**, **polymorphism**, etc.

**Simula** is considered the first object-oriented programming language. The programming paradigm where everything is represented as an object is known as a truly object-oriented programming language.

**Smalltalk** is considered the first truly object-oriented programming language.

The popular object-oriented languages are [Java](https://www.javatpoint.com/java-tutorial), [C#](https://www.javatpoint.com/c-sharp-tutorial), [PHP](https://www.javatpoint.com/php-tutorial), [Python](https://www.javatpoint.com/python-tutorial), [C++](https://www.javatpoint.com/cpp-tutorial), etc.

The main aim of object-oriented programming is to implement real-world entities, for example, object, classes, abstraction, inheritance, polymorphism, etc.

## OOPs (Object-Oriented Programming System)

**Object** means a real-world entity such as a pen, chair, table, computer, watch, etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

* [Object](https://www.javatpoint.com/object-and-class-in-java)
* Class
* [Inheritance](https://www.javatpoint.com/inheritance-in-java)
* [Polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java)
* [Abstraction](https://www.javatpoint.com/abstract-class-in-java)
* [Encapsulation](https://www.javatpoint.com/encapsulation)

Apart from these concepts, there are some other terms which are used in Object-Oriented design:

* Coupling
* Cohesion
* Association
* Aggregation
* Composition



## Object



Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.

An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.

**Example:** A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

## Class

*Collection of objects* is called class. It is a logical entity.

A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

### Inheritance

*When one object acquires all the properties and behaviors of a parent object*, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.



### Polymorphism

If *one task is performed in different ways*, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.

In Java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something; for example, a cat speaks meow, dog barks woof, etc.

#### Abstraction

*Hiding internal details and showing functionality* is known as abstraction. For example phone call, we don't know the internal processing.

In Java, we use abstract class and interface to achieve abstraction.



### Encapsulation

*Binding (or wrapping) code and data together into a single unit are known as encapsulation*. For example, a capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

### Coupling

Coupling refers to the knowledge or information or dependency of another class. It arises when classes are aware of each other. If a class has the details information of another class, there is strong coupling. In Java, we use private, protected, and public modifiers to display the visibility level of a class, method, and field. You can use interfaces for the weaker coupling because there is no concrete implementation.

### Cohesion

Cohesion refers to the level of a component which performs a single well-defined task. A single well-defined task is done by a highly cohesive method. The weakly cohesive method will split the task into separate parts. The java.io package is a highly cohesive package because it has I/O related classes and interface. However, the java.util package is a weakly cohesive package because it has unrelated classes and interfaces.

### Association

Association represents the relationship between the objects. Here, one object can be associated with one object or many objects. There can be four types of association between the objects:

* One to One
* One to Many
* Many to One, and
* Many to Many

Let's understand the relationship with real-time examples. For example, One country can have one prime minister (one to one), and a prime minister can have many ministers (one to many). Also, many MP's can have one prime minister (many to one), and many ministers can have many departments (many to many).

Association can be undirectional or bidirectional.

### Aggregation

Aggregation is a way to achieve Association. Aggregation represents the relationship where one object contains other objects as a part of its state. It represents the weak relationship between objects. It is also termed as a *has-a* relationship in Java. Like, inheritance represents the *is-a* relationship. It is another way to reuse objects.

### Composition

The composition is also a way to achieve Association. The composition represents the relationship where one object contains other objects as a part of its state. There is a strong relationship between the containing object and the dependent object. It is the state where containing objects do not have an independent existence. If you delete the parent object, all the child objects will be deleted automatically.

## Advantage of OOPs over Procedure-oriented programming language

1) OOPs makes development and maintenance easier, whereas, in a procedure-oriented programming language, it is not easy to manage if code grows as project size increases.

2) OOPs provides data hiding, whereas, in a procedure-oriented programming language, global data can be accessed from anywhere.

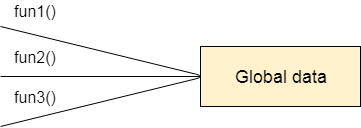


Figure: Data Representation in Procedure-Oriented Programming

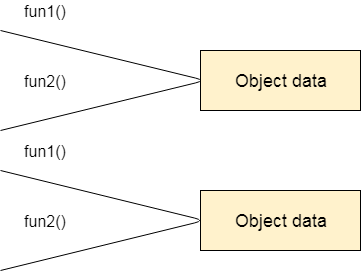
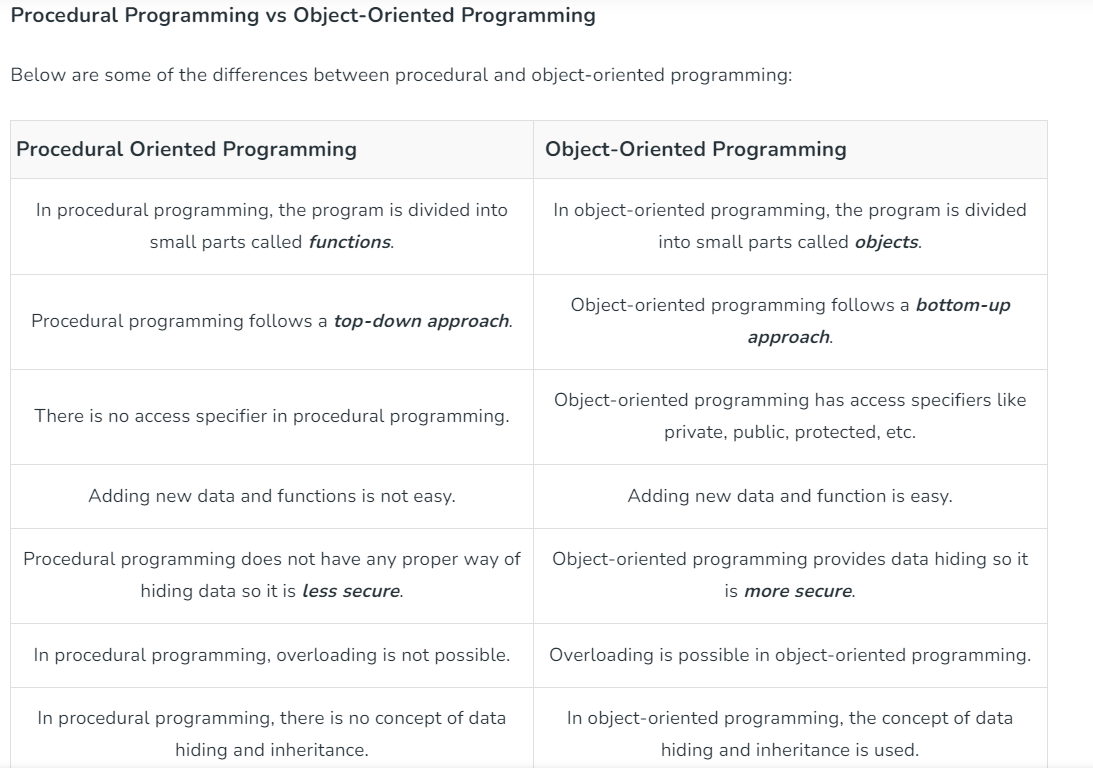


Figure: Data Representation in Object-Oriented Programming

3) OOPs provides the ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language.



**Benefits of OOP include**

* **Modularity.** Encapsulation enables objects to be self-contained, making troubleshooting and collaborative development easier.
* **Reusability.** Code can be reused through inheritance, meaning a team does not have to write the same code multiple times.
* **Productivity.** Programmers can construct new programs quicker through the use of multiple libraries and reusable code.
* **Easily upgradable and scalable.**Programmers can implement system functionalities independently.
* **Interface descriptions.** Descriptions of external systems are simple, due to message passing techniques that are used for objects communication.
* **Security.** Using encapsulation and abstraction, complex code is hidden, software maintenance is easier and [internet protocols](https://www.techtarget.com/searchunifiedcommunications/definition/Internet-Protocol) are protected.
* **Flexibility.** Polymorphism enables a single function to adapt to the class it is placed in. Different objects can also pass through the same interface.

## Applications of Object-Oriented Programming

**1. Client-Server Systems**

Object-oriented client-server systems provide the IT infrastructure, creating Object-Oriented Client-Server Internet (OCSI) applications. Here, infrastructure refers to operating systems, networks, and hardware. OSCI consist of three major technologies:

* The Client Server
* Object-Oriented Programming
* The Internet

**2. Object-Oriented Databases**

They are also called Object Database Management Systems (ODBMS). These databases store objects instead of data, such as real numbers and integers. Objects consist of the following:

**Attributes:**Attributes are data that define the traits of an object. This data can be as simple as integers and real numbers. It can also be a reference to a complex object.

**Methods:**They define the behavior and are also called functions or procedures.

**3. Object-Oriented Databases**

These databases try to maintain a direct correspondence between the real-world and database objects in order to let the object retain its identity and integrity. They can then be identified and operated upon.

**4. Real-Time System Design**

Real-time systems inherent complexities that make it difficult to build them. Object-oriented techniques make it easier to handle those complexities. These techniques present ways of dealing with these complexities by providing an integrated framework, which includes schedulability analysis and behavioral specifications.

**5. Simulation and Modeling System**

It’s difficult to model complex systems due to the varying specification of variables. These are prevalent in medicine and in other areas of natural science, such as ecology, zoology, and agronomic systems.  Simulating complex systems requires modeling and understanding interactions explicitly. Object-oriented programming provides an alternative approach for simplifying these complex modeling systems.

**6. Hypertext and Hypermedia**

OOP also helps in laying out a framework for hypertext. Basically, hypertext is similar to regular text, as it can be stored, searched, and edited easily. The only difference is that hypertext is text with pointers to other text as well.

Hypermedia, on the other hand, is a superset of hypertext. Documents having hypermedia not only contain links to other pieces of text and information but also to numerous other forms of media, ranging from images to sound.

**7. Neural Networking and Parallel Programming**

It addresses the problem of prediction and approximation of complex time-varying systems. Firstly, the entire time-varying process is split into several time intervals or slots. Then, neural networks are developed in a particular time interval to disperse the load of various networks. OOP simplifies the entire process by simplifying the approximation and prediction ability of networks.

**8. Office Automation Systems**

These include formal as well as informal electronic systems primarily concerned with information sharing and communication to and from people inside and outside the organization. Some examples are:

* Email
* Word processing
* Web calendars
* Desktop publishing

**9. CIM/CAD/CAM Systems**

OOP can also be used in manufacturing and design applications, as it allows people to reduce the effort involved. For instance, it can be used while designing blueprints and flowcharts. OOP makes it possible for the designers and engineers to produce these flowcharts and blueprints accurately.

**10. AI Expert Systems**

These are computer applications that are developed to solve complex problems pertaining to a specific domain, which is at a level far beyond the reach of a human brain.

It has the following characteristics:

* Reliable
* Highly responsive
* Understandable

**OBJECTS : An object in Java is the physical as well as a logical entity, whereas, a class in Java is a logical entity only.**

### **What is an object in Java**

An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical (tangible and intangible). The example of an intangible object is the banking system.

An object has three characteristics:

* **State:** represents the data (value) of an object.
* **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.
* **Identity:** An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

For Example, Pen is an object. Its name is Reynolds; color is white, known as its state. It is used to write, so writing is its behavior.

**An object is an instance of a class.** A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

**Object Definitions:**

* An object is *a real-world entity*.
* An object is *a runtime entity*.
* The object is *an entity which has state and behavior*.
* The object is *an instance of a class*.

## class in Java

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

* **Fields**
* **Methods**
* **Constructors**
* **Blocks**
* **Nested class and interface**

# **Abstraction Vs Encapsulation**

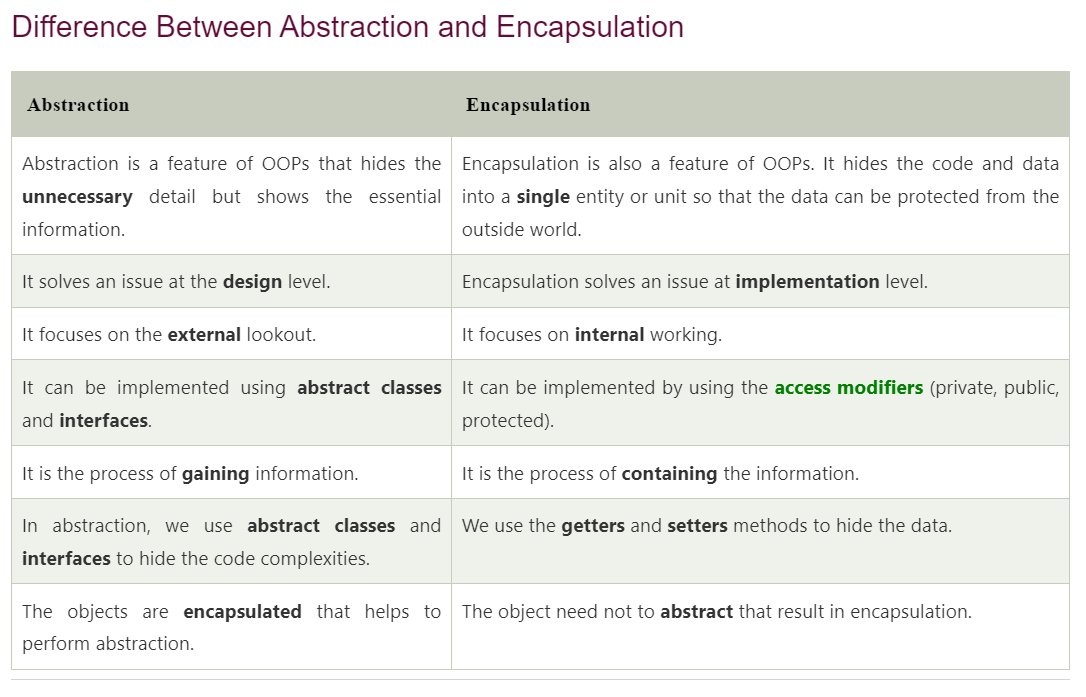
Java is an object-oriented programming language and it follows [OOPs concepts](https://www.javatpoint.com/java-oops-concepts). The OOPs concepts include [classes, objects,](https://www.javatpoint.com/object-and-class-in-java) polymorphism, [inheritance](https://www.javatpoint.com/inheritance-in-java). There are two other features of OOPs i.e. [abstraction](https://www.javatpoint.com/abstract-class-in-java) and [encapsulation](https://www.javatpoint.com/encapsulation). They both seem very similar but totally different in concept and implementation. The **major difference between abstraction and encapsulation** is that abstraction hides the code complexity while encapsulation hides the internal working from the outside world. In this section, we will discuss abstraction and encapsulation and the **differences between abstraction and encapsulation in Java**.

## Abstraction

It is a feature of OOPs. It is used to hide the unnecessary information or data from the user but shows the essential data that is useful for the user. It can be achieved by using the [interface](https://www.javatpoint.com/interface-in-java) and the [abstract class](https://www.javatpoint.com/abstract-class-in-java). In interfaces, only the methods are exposed to the end-user. The best example of abstraction is a **TV remote**. The user only interacts with the outer interface that is nothing but keys. The user only knows which key to press for what function.

## Encapsulation

It is also a feature of OOP. It is used to bind up the data into a single unit called class. It provides the mechanism which is known as **data hiding**. It is an important feature of OOPs. It prevents to access data members from the outside of the class. It is also necessary from the security point of view.



# **Inheritance in Java**

**Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of [OOPs](https://www.javatpoint.com/java-oops-concepts) (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new [classes](https://www.javatpoint.com/object-and-class-in-java) that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the **IS-A relationship** which is also known as a parent-child relationship.

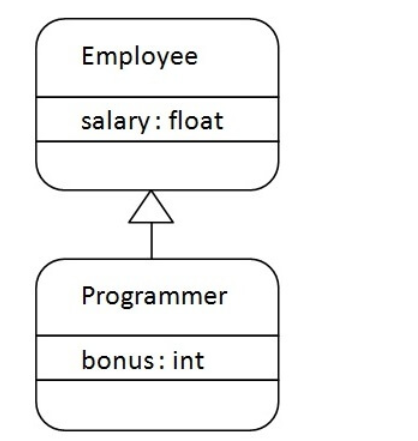
### **Why use inheritance in java**

* For Method Overriding (so runtime polymorphism can be achieved).
* For Code Reusability.

### **Terms used in Inheritance**

* **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
* **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
* **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
* **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

### **Java Inheritance Example**



A diagram of a program

Description automatically generated

A diagram of a group of objects

Description automatically generated

**OVERLOADING and OVERRIDING**

In Java, method overloading and method overriding both refer to creating different methods that share the same name.

While the two concepts share some similarities, they are distinct notions with markedly different use cases. Having a firm grasp of them is important in building strong foundational Java skills.

In this post, we'll explore the key rules of method overloading and overriding, as well as the differences between them.

## What is Method Overloading in Java?

Overloading a method, in simple terms, means creating a different method with the same name in the same class, but with a different parameter list.

There can be many cases where you might need to handle different types of input for the same operation, and method overloading is one way to handle such cases.

For example, let's say you want to create a method that performs an addition of two numbers. This calculation is designed to return a number as its output. If your method handles parameters of type int, attempting to call it by passing values of type double as arguments results in a compilation error.

For this reason, you might want to overload the method by creating a new version of that method that is able to handle a different type of input (in this case of type double):

public class Calculator {

public int sum(int a, int b) {

return a + b;

}

public double sum(double a, double b) {

return a + b;

}

}

In the example above, the sum() method is overloaded, because it is defined more than once within the same class, but with a different parameter list.

A method can also be overloaded by changing the number of parameters. On this basis, the following methods are also legal examples of how the sum() method can be overloaded, assuming they are placed within the same class:

public int sum(int a, int b, int c) {

return a + b + c;

}

protected void sum() {

System.out.print("Nothing to sum");

}

Note that, as in some of the examples above, you can also change the return type or the access modifier, but this is not mandatory.

### Key Rules of Method Overloading

Remember these rules when overloading a method:

* The overloaded and overloading methods must be in the same class (Note: this includes any methods inherited, even implicitly, from a superclass).
* The method parameters must change: either the number or the type of parameters must be different in the two methods.
* The return type can be freely modified.
* The access modifier (public, private, and so on) can be freely modified.
* Thrown exceptions, if any, can be freely modified.

## What is Method Overriding in Java?

Method overriding refers to redefining a method in a subclass that already exists in the superclass.

When you call an overridden method using an object of the subclass type, Java uses the method's implementation in the subclass rather than the one in the superclass. For this reason, an understanding of the concept of inheritance in Java is important in order to get a good grasp of method overriding.

Any subclass can generally override any method from a superclass, unless a method is marked with the final or static keywords. The overriding method must not change the name and parameter list of the overridden method.

While not compulsory, it is good practice to use the @Override annotation when overriding a method: this annotation will check that the method is being overridden correctly, and will warn you if that's not the case.

In the following example, you'll see a class Car that extends the class Vehicle. The Car class overrides the move() method from the superclass, and this is made explicit by the use of the @Override annotation. The two methods are implemented differently in the method body.

class Vehicle {

public void move() {

System.out.println("The vehicle is moving");

}

}

class Car extends Vehicle {

@Override

public void move() {

System.out.println("The car is moving");

}

}

The choice of which version of move() will be called is based on the object type the method is being called on. Note that the version of the overridden method that is called is determined at runtime and is based on the object type, not the object reference.

This is illustrated in the following example, particularly in the third call to move(): while the method is called on an object reference on type Vehicle, the actual object is of type Car. The type of the object here is determined at runtime, and the version of the method that is called is therefore the one from the Car subclass.

public static void main(String[] args) {

Vehicle vehicle = new Vehicle();

vehicle.move(); // Prints: The vehicle is moving

Car car = new Car();

car.move(); // Prints: The car is moving

Vehicle secondVehicle = new Car();

secondVehicle.move(); // Prints: The car is moving

}

### Key Rules of Method Overriding

Remember these rules when overriding a method:

* The parameter list must not change: the overriding method must take the same number and type of parameters as the overridden method – otherwise, you would just be overloading the method.
* The return type must not change (Note: if the method returns an object, a subclass of that object is allowed as the return type).
* The access modifier must be either the same or a less restrictive one (for example, if the overridden method is protected, you can declare the overriding method as public, but not private).
* Thrown checked exceptions, if any, can be removed or reduced by the overriding method. This means that the overriding method can throw the same checked exception as the overridden method, or a subclass of that checked exception, but not a broader exception. This restriction does not apply to unchecked exceptions.

**JVM -Java Virtual Machine**

**Java Expressions**

A Java expression consists of variables, operators, literals, and method calls. To know more about method calls, visit Java methods. For example,

int score;

score = 90;

Here, score = 90 is an expression that returns an int.

Consider another example,

Double a = 2.2, b = 3.4, result;

result = a + b - 3.4;

Here, a + b - 3.4 is an expression.

if (number1 == number2)

System.out.println("Number 1 is larger than number 2");

Here, number1 == number2 is an expression that returns a boolean value. Similarly, "Number 1 is larger than number 2" is a string expression.

---------------------------------------------------------------------------------------

**Java Statements**

In Java, each statement is a complete unit of execution.

For example,

int score = 9\*5;

Here, we have a statement. The complete execution of this statement involves multiplying integers 9 and 5 and then assigning the result to the variable score.

In the above statement, we have an expression 9 \* 5.

In Java, expressions are part of statements.

**Expression statements**

We can convert an expression into a statement by terminating the expression with a ;. These are known as expression statements. For example,

// expression

number = 10

// statement

number = 10;

In the above example, we have an expression number = 10. Here, by adding a semicolon (;), we have converted the expression into a statement (number = 10;).

Consider another example,

// expression

++number

// statement

++number;

Similarly, ++number is an expression whereas ++number; is a statement.

**Declaration Statements**

In Java, declaration statements are used for declaring variables. For example,

Double tax = 9.5;

The statement above declares a variable tax which is initialized to 9.5.

Note: There are control flow statements that are used in decision making and looping in Java.

**Java Blocks**

A block is a group of statements (zero or more) that is enclosed in curly braces { }.

For example,

class Main {

public static void main(String[] args) {

String band = "Beatles";

if (band == "Beatles") { // start of block

System.out.print("Hey ");

System.out.print("Jude!");

} // end of block

}

}

Output:

Hey Jude!

In the above example, we have a block if {....}.

Here, inside the block we have two statements:

System.out.print("Hey ");

System.out.print("Jude!");

However, a block may not have any statements.

Consider the following examples,

class Main {

public static void main(String[] args) {

if (10 > 5) { // start of block

} // end of block }

}

This is a valid Java program. Here, we have a block if {...}. However, there is no any statement inside this block.

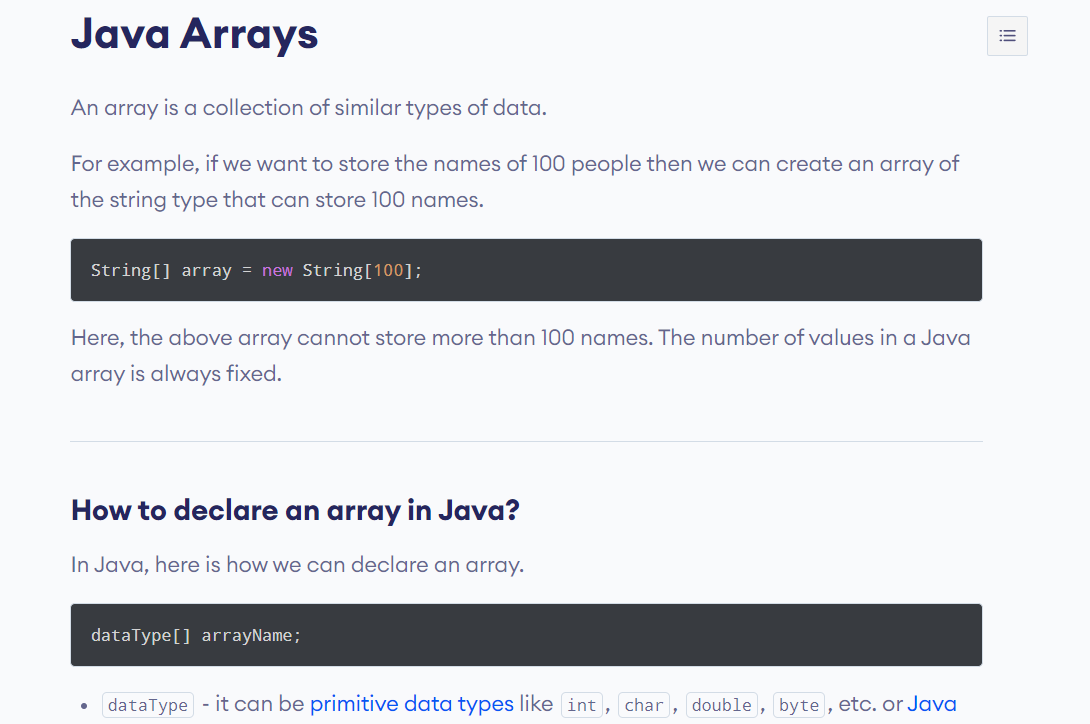
class AssignmentOperator {

public static void main(String[] args) { // start of block

} // end of block

}

Here, we have block public static void main() {...}. However, similar to the above example, this block does not have any statement.



A screenshot of a computer

Description automatically generated

A screenshot of a computer

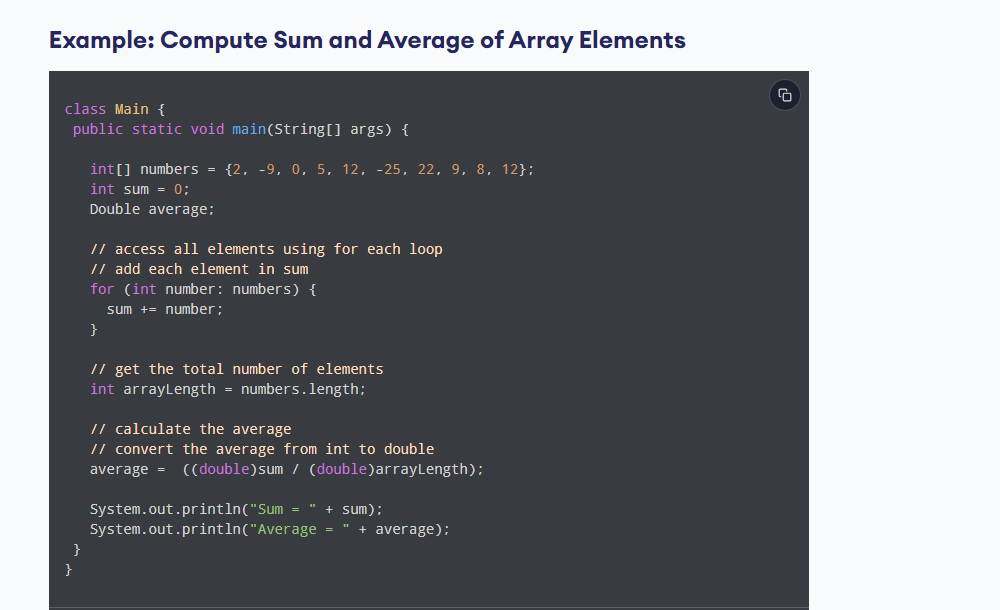
Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated



A screenshot of a computer

Description automatically generated

## What is a Constructor?

A constructor in Java is similar to a method that is invoked when an object of the class is created.

Unlike **java methods** a constructor has the same name as that of the class and does not have any return type. For example,

class Test {

Test() {

// constructor body

}

}

Here, Test() is a constructor. It has the same name as that of the class and doesn't have a return type.

## Example 1: Java Constructor

class Main {

private String name;

// constructor

Main() {

System.out.println("Constructor Called:");

name = "Programiz";

}

public static void main(String[] args) {

// constructor is invoked while

// creating an object of the Main class

Main obj = new Main();

System.out.println("The name is " + obj.name);

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**:

Constructor Called:

The name is Programiz

In the above example, we have created a constructor named Main(). Inside the constructor, we are initializing the value of the name variable.

Notice the statement of creating an object of the Main class.

Main obj = new Main();

Here, when the object is created, the Main() constructor is called. And, the value of the name variable is initialized.

Hence, the program prints the value of the name variables as Programiz.

### **Types of Constructor**

In Java, constructors can be divided into 3 types:

1. No-Arg Constructor
2. Parameterized Constructor
3. Default Constructor

## 1. Java No-Arg Constructors

Similar to methods, a Java constructor may or may not have any parameters (arguments).If a constructor does not accept any parameters, it is known as a no-argument constructor. For example,

private Constructor() {

// body of the constructor

}

### Example 2: Java private no-arg constructor

class Main {

int i;

// constructor with no parameter

private Main() {

i = 5;

System.out.println("Constructor is called");

}

public static void main(String[] args) {

// calling the constructor without any parameter

Main obj = new Main();

System.out.println("Value of i: " + obj.i);

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**:

Constructor is called

Value of i: 5

In the above example, we have created a constructor Main(). Here, the constructor does not accept any parameters. Hence, it is known as a no-arg constructor.

**Notice that we have declared the constructor as private.**

Once a constructor is declared private, it cannot be accessed from outside the class. So, creating objects from outside the class is prohibited using the private constructor.

Here, we are creating the object inside the same class. Hence, the program is able to access the constructor.

However, if we want to create objects outside the class, then we need to declare the constructor as public.

### Example 3: Java public no-arg constructors

class Company {

String name;

// public constructor

public Company() {

name = "Programiz";

}

}

class Main {

public static void main(String[] args) {

// object is created in another class

Company obj = new Company();

System.out.println("Company name = " + obj.name);

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**:

Company name = Programiz

## 2. Java Parameterized Constructor

A Java constructor can also accept one or more parameters. Such constructors are known as parameterized constructors (constructor with parameters).

### Example 4: Parameterized constructor

class Main {

String languages;

// constructor accepting single value

Main(String lang) {

languages = lang;

System.out.println(languages + " Programming Language");

}

public static void main(String[] args) {

// call constructor by passing a single value

Main obj1 = new Main("Java");

Main obj2 = new Main("Python");

Main obj3 = new Main("C");

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**:

Java Programming Language

Python Programming Language

C Programming Language

In the above example, we have created a constructor named Main(). Here, the constructor takes a single parameter. Notice the expression,

Main obj1 = new Main("Java");

Here, we are passing the single value to the constructor. Based on the argument passed, the language variable is initialized inside the constructor.

## 3. Java Default Constructor

If we do not create any constructor, the Java compiler automatically create a no-arg constructor during the execution of the program. This constructor is called default constructor.

### Example 5: Default Constructor

class Main {

int a;

boolean b;

public static void main(String[] args) {

// A default constructor is called

Main obj = new Main();

System.out.println("Default Value:");

System.out.println("a = " + obj.a);

System.out.println("b = " + obj.b);

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**:

Default Value:

a = 0

b = false

Here, we haven't created any constructors. Hence, the Java compiler automatically creates the default constructor.

The default constructor initializes any uninitialized instance variables with default values.

|  |  |
| --- | --- |
| Type | Default Value |
| Boolean | false |
| Byte | 0 |
| Short | 0 |
| Int | 0 |
| Long | 0L |
| Char | \u0000 |
| Float | 0.0f |
| Double | 0.0d |
| Object | Reference null |

In the above program, the variables a and b are initialized with default value **0** and false respectively.

The above program is equivalent to:

class Main {

int a;

boolean b;

Main() {

a = 0;

b = false;

}

public static void main(String[] args) {

// call the constructor

Main obj = new Main();

System.out.println("Default Value:");

System.out.println("a = " + obj.a);

System.out.println("b = " + obj.b);

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**The output of the program is the same as Example 5.**

## Important Notes on Java Constructors

* Constructors are invoked implicitly when you instantiate objects.
* The two rules for creating a constructor are:  
  The name of the constructor should be the same as the class.  
  A Java constructor must not have a return type.
* If a class doesn't have a constructor, the Java compiler automatically creates a **default constructor** during run-time. The default constructor initializes instance variables with default values. For example, the int variable will be initialized to 0
* Constructor types:  
  **No-Arg Constructor** - a constructor that does not accept any arguments  
  **Parameterized constructor** - a constructor that accepts arguments  
  **Default Constructor** - a constructor that is automatically created by the Java compiler if it is not explicitly defined.
* A constructor cannot be abstract or static or final.
* A constructor can be overloaded but can not be overridden.