★ Structured Approach Vs. Object-Oriented Approach

Structured Approach	Object-Oriented Approach
Program is divided into a number of sub modules or functions.	Program is organized by having a number of classes and objects.
Function call is used.	Message passing is used.
Problem oriented approach	Problem domain oriented approach
Software reuse is not possible.	Reusability is possible.

★ UML (Unified Modeling Language)

UML is a standard modeling language for specifying, visualizing, constructing, and documenting the artifacts of systems.UML is a pictorial language used to make system blueprints.

There are two main categories;

- structure diagrams

  Structure diagrams show the things in the modeled system. In a more technical term, they show different objects in a system.
  1. Class Diagram
  2. Component Diagram
  3. Deployment Diagram
  4. Object Diagram
  5. Package Diagram
  6. Profile Diagram
  7. Composite Structure Diagram

- behavioral diagrams.

  show what should happen in a system. They describe how the objects interact with each other to create a functioning system.
  1. Use Case Diagram
  2. Activity Diagram
  3. State Machine Diagram
  4. Sequence Diagram
  5. Communication Diagram
  6. Interaction Overview Diagram
  7. Timing Diagram

structure diagrams

1. **Class diagram**

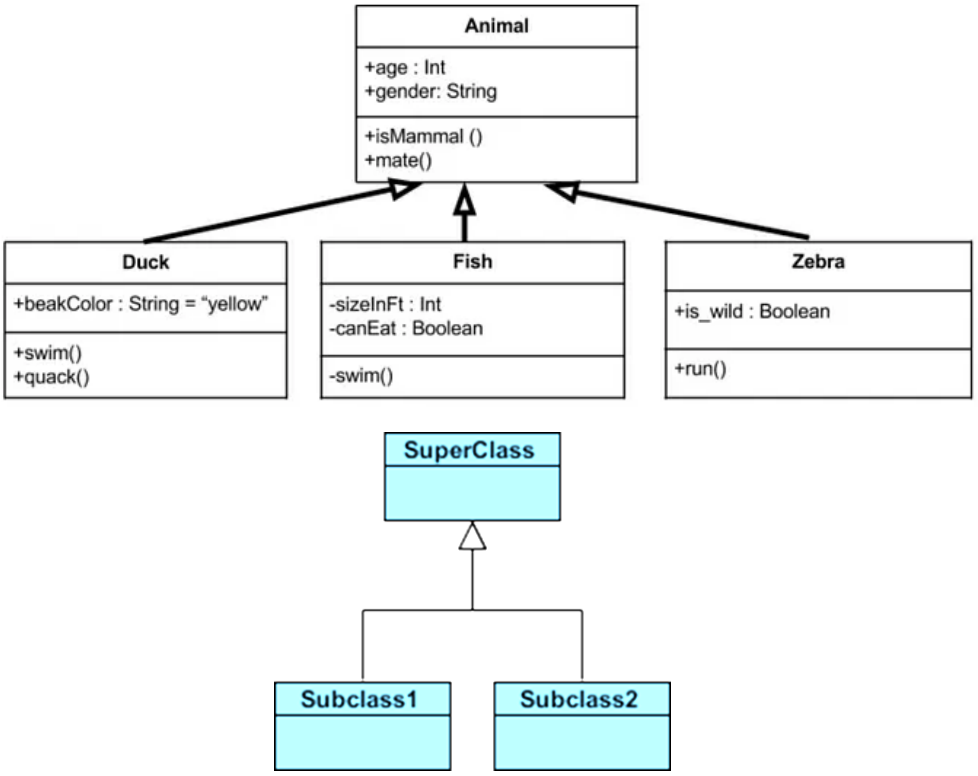
It describes the objects in a system and their relationships. Class diagram consists of attributes and functions. Class diagrams are the only UML diagrams which can be mapped directly with object oriented languages.

+ denotes - public attributes or operations
- denotes - private attributes or operations
 # denotes - protected attributes or operations
~ denotes - package attributes or operations

Nortation / Class Relationships

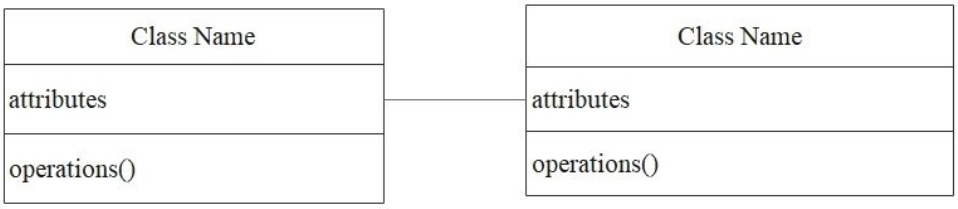
1. Inheritance (or Generalization)

Inheritance is “is a relationship”. It has a parent class and its corresponding child classes



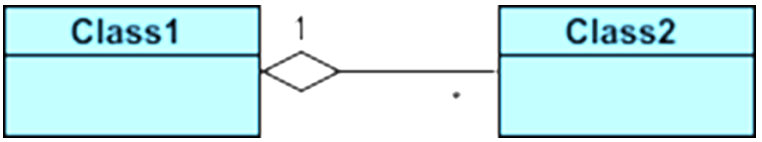
2. Simple Association

A structural link between two peer classes.There is an association between Class1 and Class2.consider we have two classes A and B where class A calls class B and Class B also calls class A.



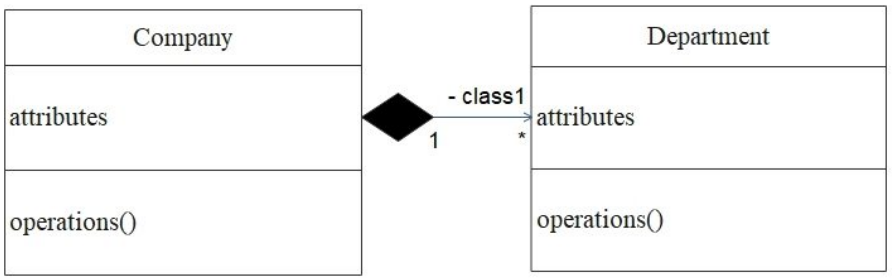
3. Aggregation

A special type of association. It represents a "part of" relationship. Class2 is part of Class1.

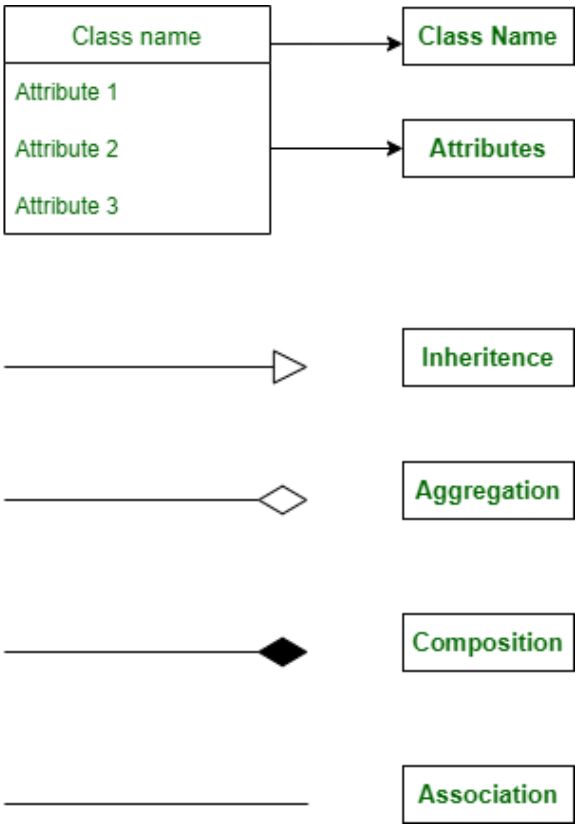


4. Composition

one class is dependent on another. One class is a part of the other.



5. Dependency



Multiplicity

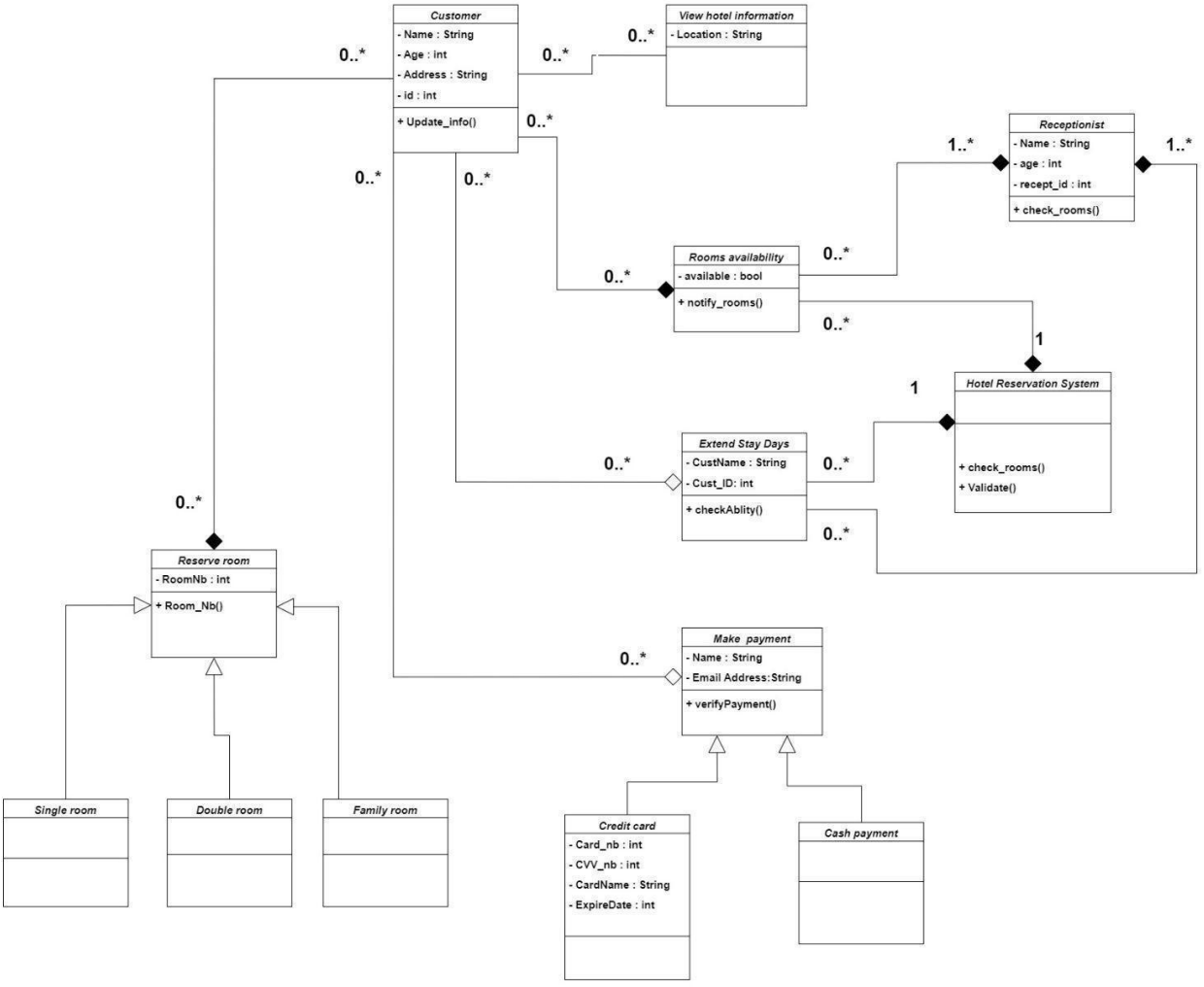
How many objects of each class take part in the relationships and multiplicity can be expressed as:

- Exactly one - 1
- Zero or one - 0..1
- Many - 0..* or *
- One or more - 1..*
- Exact Number - e.g. 3..4 or 6
- Or a complex relationship - e.g. 0..1, 3..4, 6.* would mean any number of objects other than 2 or 5

Question 1 -

Draw a class diagram for the following scenario in a hotel The following text describes the proposed Hotel Reservation System.

A customer can view the hotel information and reserve a room online or visit the place and reserve rooms. Rooms can be Single rooms, double rooms, or family rooms.If they are online customers, the system will check the room availability and notify the customer availability of rooms. If not, the receptionist does the task.When the customer makes the payment, the room is allocated for them. Online customers should pay via credit cards and others are allowed to pay by credit cards or cash payment.When the customer checks in the system is updated. The customer can check out from the hotel at any time or extend after informing the receptionist. It will be accepted if rooms are available



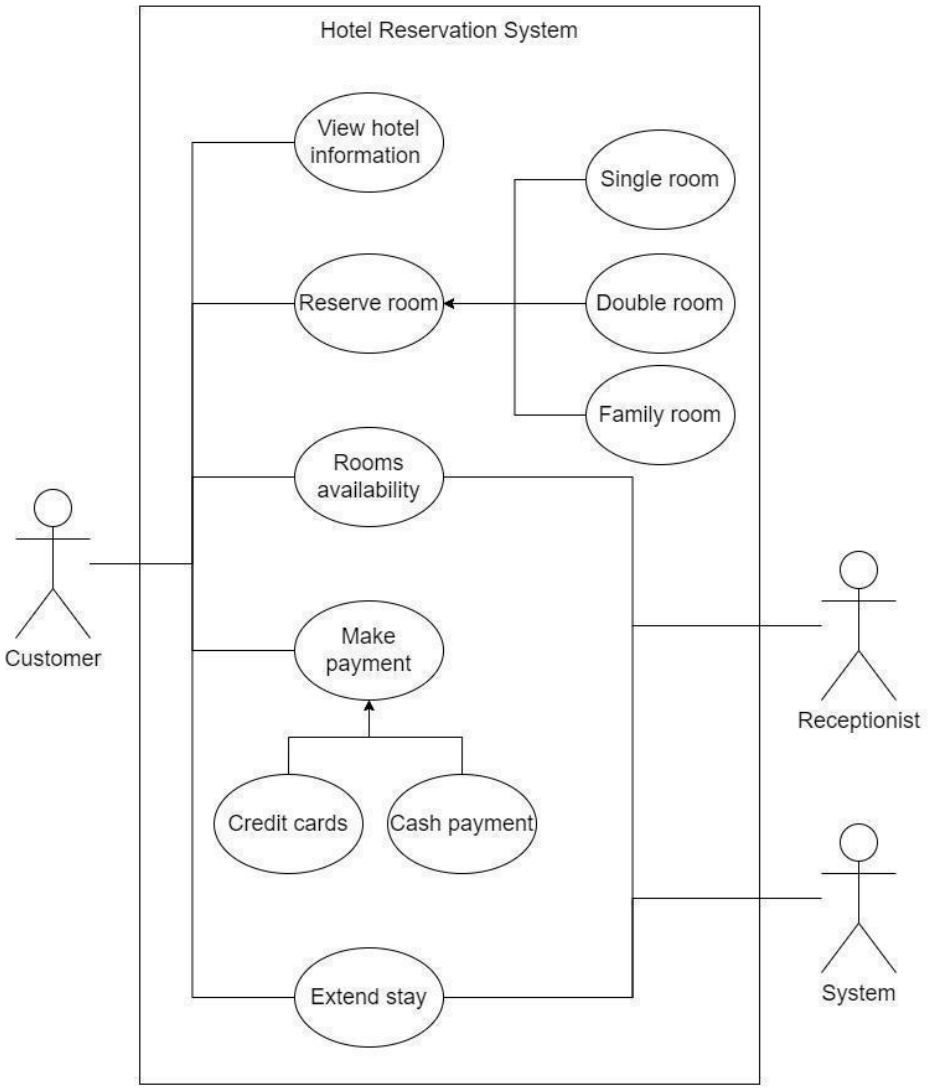
Use case diagram

It consists of use cases, actors and their relationships. Use case diagrams are used at a high level design to capture the requirements of a system. So it represents the system functionalities and their flow.

Purpose of USE CASE diagrams

- Representing the goals of system-user interactions
- Defining and organizing functional requirements in a system
- Specifying the context and requirements of a system
- Modeling the basic flow of events in a use case

Above question use case diagram



Sequence Diagrams

Sequence DiagramsUML Sequence Diagrams are interaction diagrams that detail how operations are carried out. They capture the interaction between objects in the context of a collaboration.

Sequence Diagrams are time focused and they show the order of the interaction visually by using the vertical axis of the diagram to represent time, what messages are sent and when.

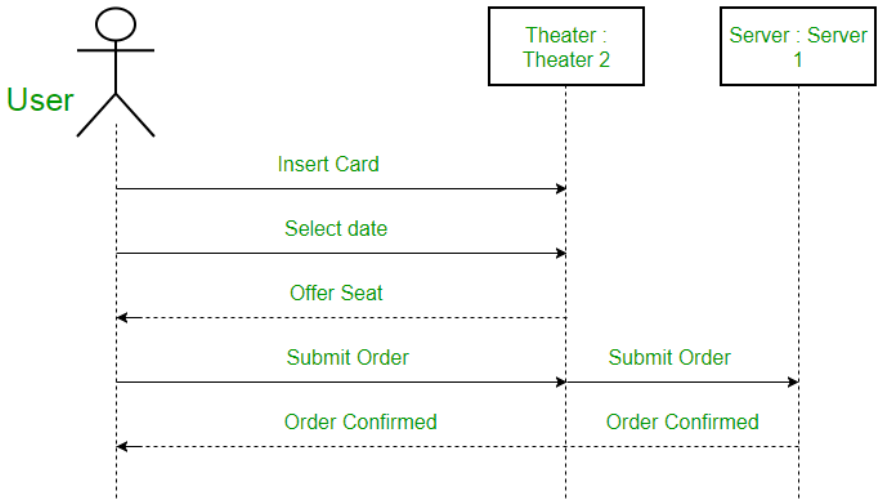
Purpose of Sequence Diagram

1. Visualize interactions between system components.
2. Understand system behavior through message sequencing.
3. Aid in system design and analysis.
4. Identify potential issues or bottlenecks.
5. Facilitate communication among stakeholders.
6. Act as valuable documentation for the system's dynamic behavior.

Notation -

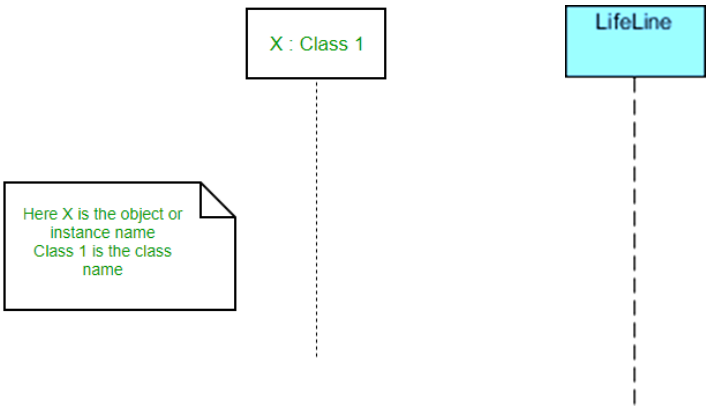
1. Actors –

An actor in a UML diagram represents a type of role where it interacts with the system and its objects.
external to the subject (i.e., in the sense that an instance of an actor is not a part of the instance of its corresponding subject).



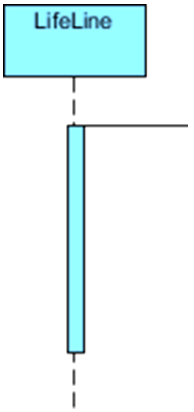
2. Lifelines –

A lifeline represents an individual participant in the Interaction



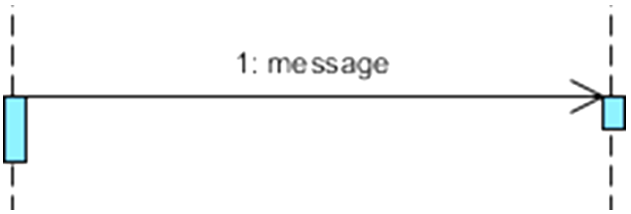
3. Activations -

A thin rectangle on a lifeline represents the period during which an element is performing an operation.
The top and the bottom of the of the rectangle are aligned with the initiation and the completion time respectively



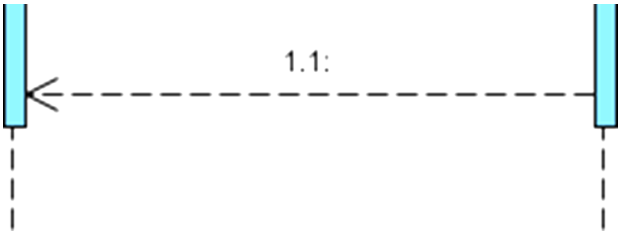
4. Call Message -

A message defines a particular communication between Lifelines of an Interaction.Call message is a kind of message that represents an invocation of operation of target lifeline.We represent messages using arrows.



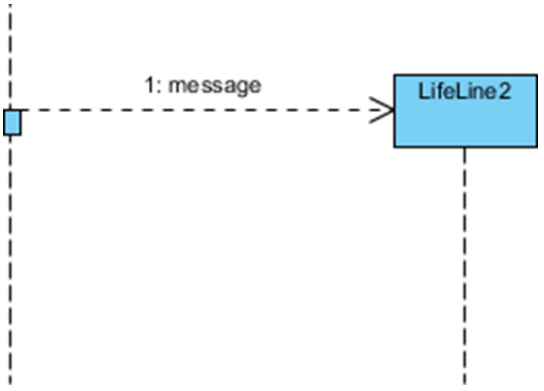
5. Return Message -

Return message is a kind of message that represents the pass of information back to the caller of a corresponding former message.

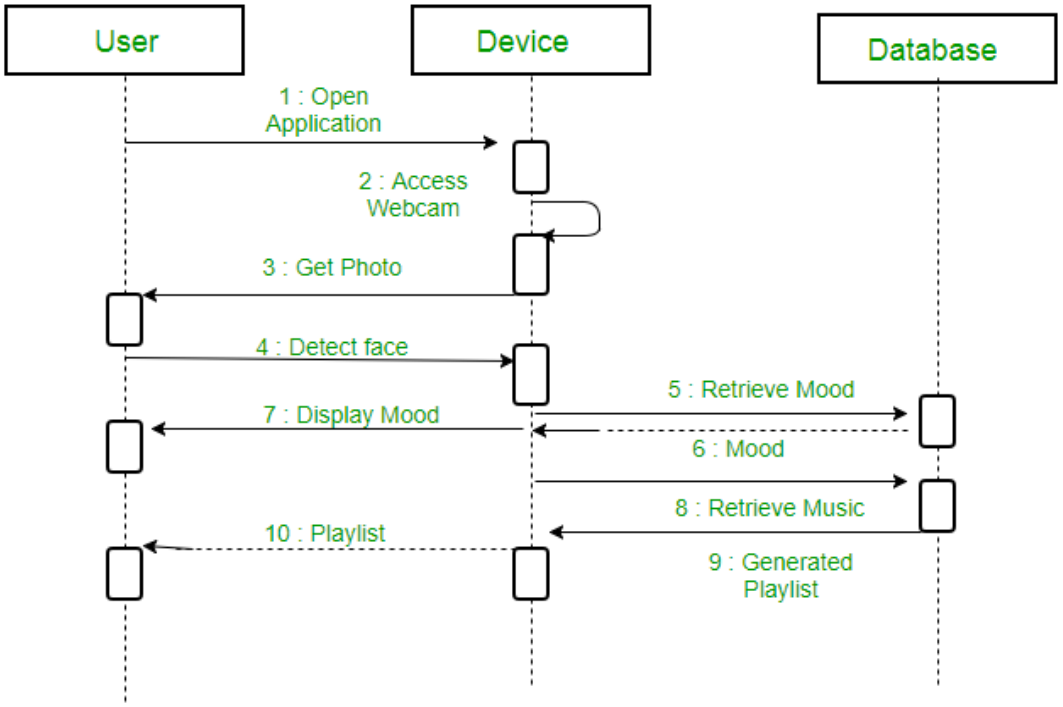


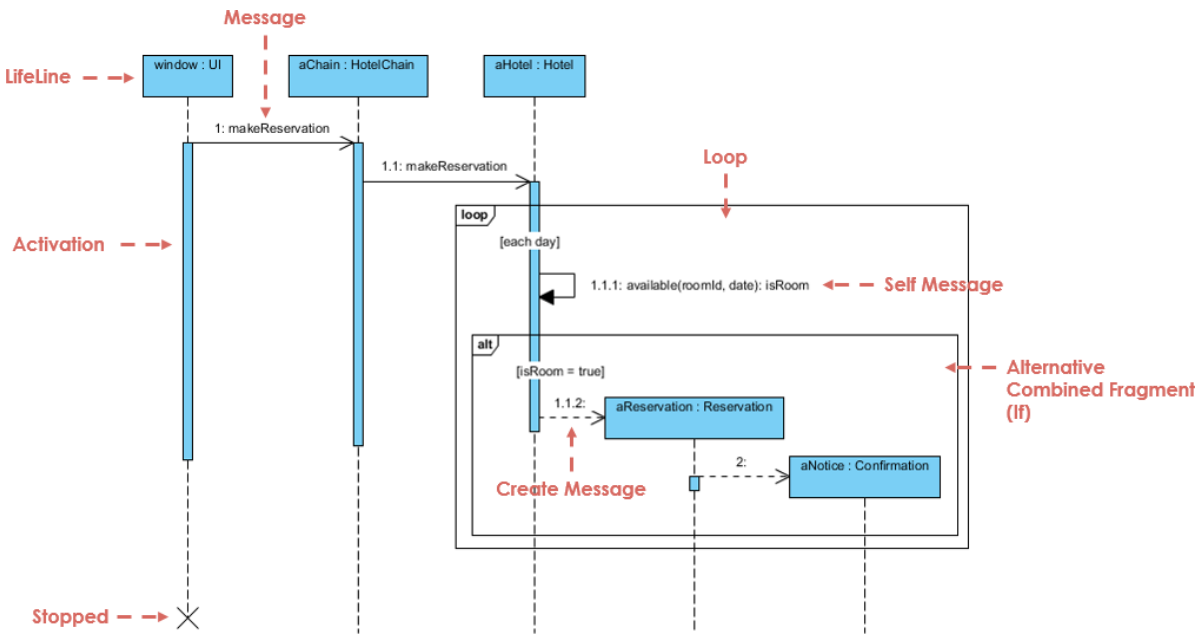
6. Create Message

Create message is a kind of message that represents the instantiation of (target) lifeline.



A sequence diagram for an emotion based music player





★ Fundamental of java

```
class HelloWorldApp
{
   public static void main(String[] args)
   {
      System.out.println("Hello World!"); // Display the text
   }
}
```

This short note making resources are,google search engine,chat gpt and lecture slides

WhatsApp group (click on this/new syllabus notes only) Credit - Kanishka viraj

- **class** keyword is used to declare a class in java.
- **public** keyword is an access modifier which represents visibility, it means it is visible to all.
- **static** is a keyword used to create a static method. no need to create object to invoke the static method
- **void** is the return type of the method
- **main** represents the startup of the program.
- **String[] args** are used for command line arguments.
- **System.out.println()** is used as a print statement.

Data types

Data types are divided into two groups:

1. Primitive data types - byte, short, int, long, float, double, boolean and char
2. Non-primitive data types - such as String, Arrays and Classes

Operators

1. Assignment Operators
2. Arithmetic Operators
3. Relational Operators / Comparison Operators
4. Bitwise Operators
5. Logical Operators

1. Arithmetic Operators

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

2. Assignment Operators

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

3. Relational Operators / Comparison Operators

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

4. Logical Operators

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

5. Bitwise Operators

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
>>	Right Shift
<<	Left Shift

★ Control flow

control flow (or flow of control) is the order in which individual statements of a program are executed

Control flow structures

1. Sequential structure :

Default mode. Statements are executed from top to bottom of the program one by one.

2. Selection structure:

Selection structure or conditional structure, is different blocks of code are performed based on whether a boolean condition is true or false

If-else, switch-case

3. Repetition structure:

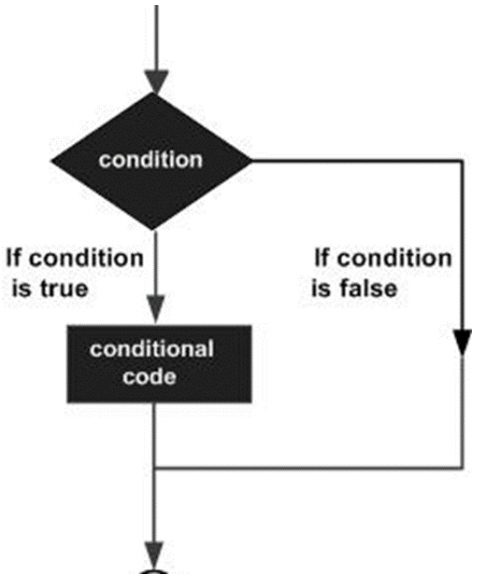
Same block of code is executed again and again based on whether a boolean condition is true or false

while, do-while, for

If condition

syntax:

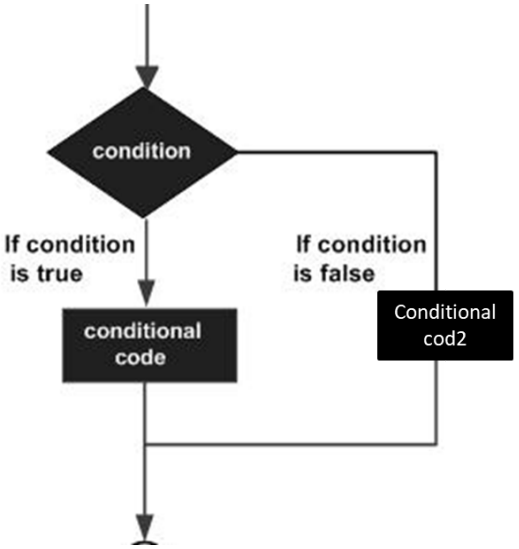
```
If(expression)
{
        statement(s)
}
```



If else condition

Syntax:

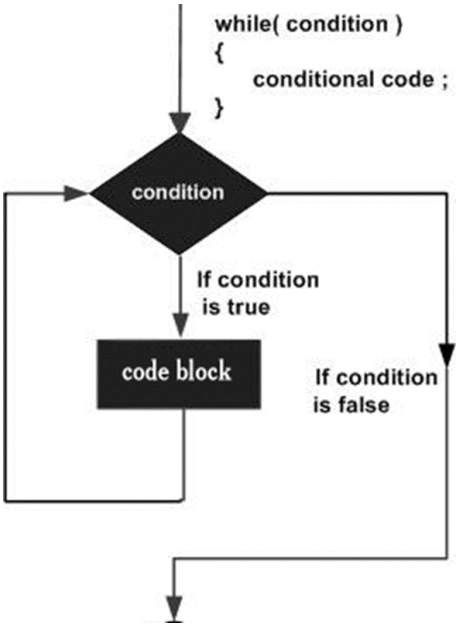
```
if (expression)
  {
      Conditional code 1
  }
Else
  {
      Conditional code 2
  }
```



while loop

Syntax:

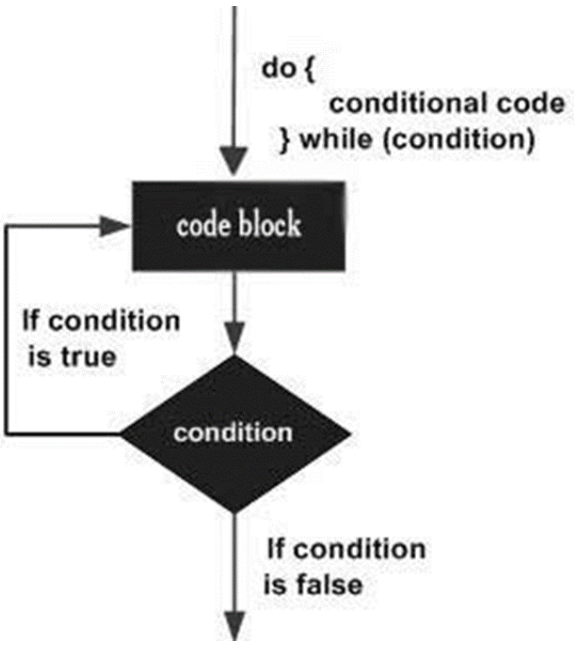
```
while (expression)
{
      statement(s)
}
```



Do While loop

Syntax:

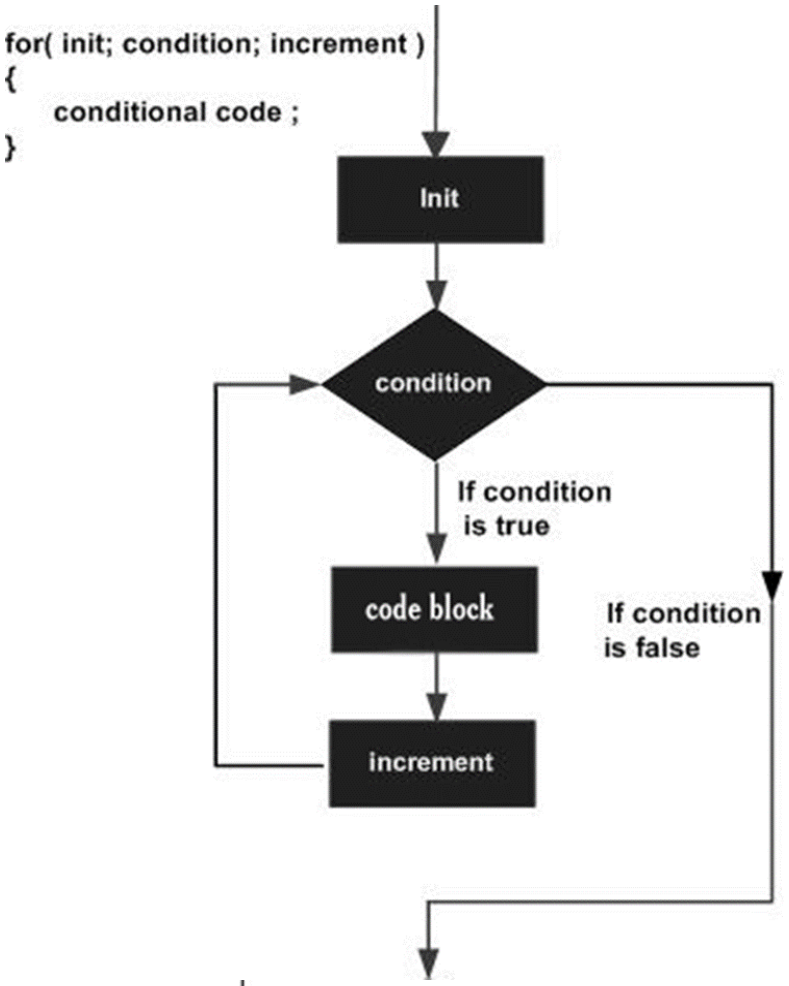
```
do
 {
      statement(s)
}
 while (expression)
```



For Loop

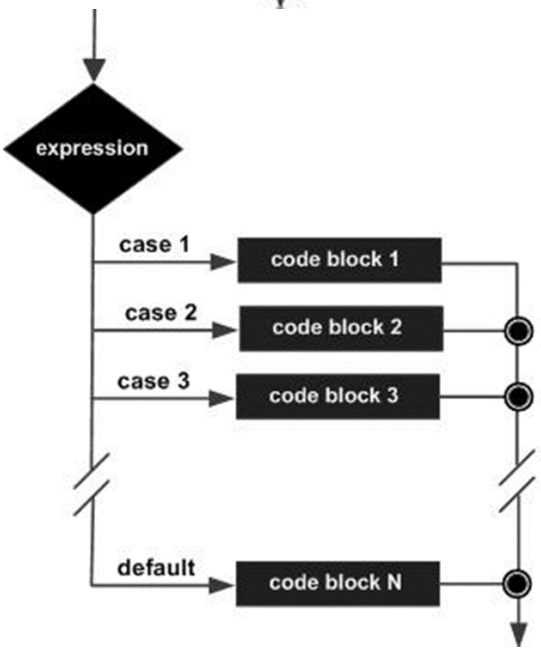
Syntax:

```
For (initialization; expression; increment / decrement)
 {
        //code block
}
```



Switch Statement

```
switch ( expression )
{
case value:
        statement(s);
       break;
case value:
        statement(s);
       break;
default:
        statement(s);
}
```



```
public class printswitch {
    public static void main(String[] args) {
        int Day = 4;
        String DayString;
                switch (Day) {
                    case 1:
                        DayString = "Sunday";
                        break;
                    case 2:
                        DayString = "Monday";
                        break;
                    case 3:
                        DayString = "Tuesday";
                        break;
                    case 4:
                        DayString = "Wednesday";
                        break;
                    case 5:
                        DayString = "Thursday";
                        break;
                    case 6:
                        DayString = "Friday";
                        break;
                    case 7:
                        DayString = "Saturday";
                        break;
                    default:
                        DayString = "Invalid Day";
                        break;
                }
        System.out.println(DayString);
    }
}
```

★ break statement

Terminates the loop or switch statement and transfers execution to the statement immediately following the loop or switch.

★ continue statement

continue statement is used in loops (such as for, while, or do-while) to skip the current iteration and move to the next one/iteration.

★ Object Oriented Programming (OOP)

Object Oriented Programming (OOP) is a programming paradigm that focuses on the use of objects to represent and manipulate data with real world concepts.

OOP concepts include,

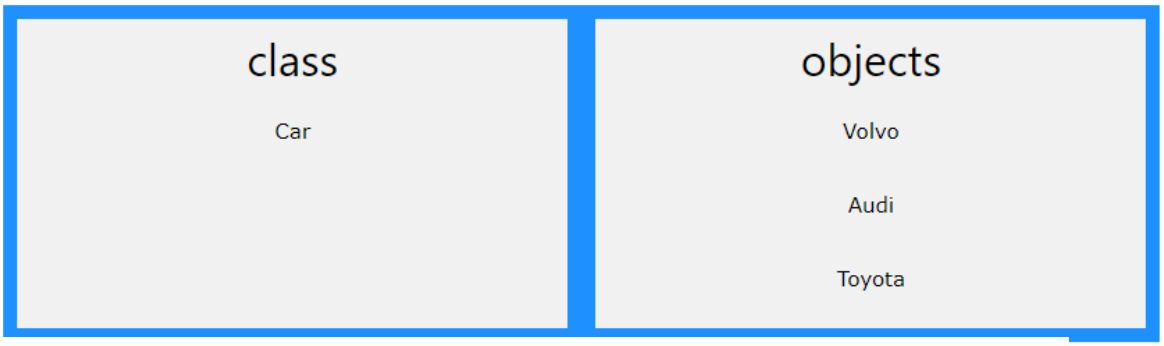
1. Inheritance
2. Abstraction
3. Encapsulation
4. Polymorphism

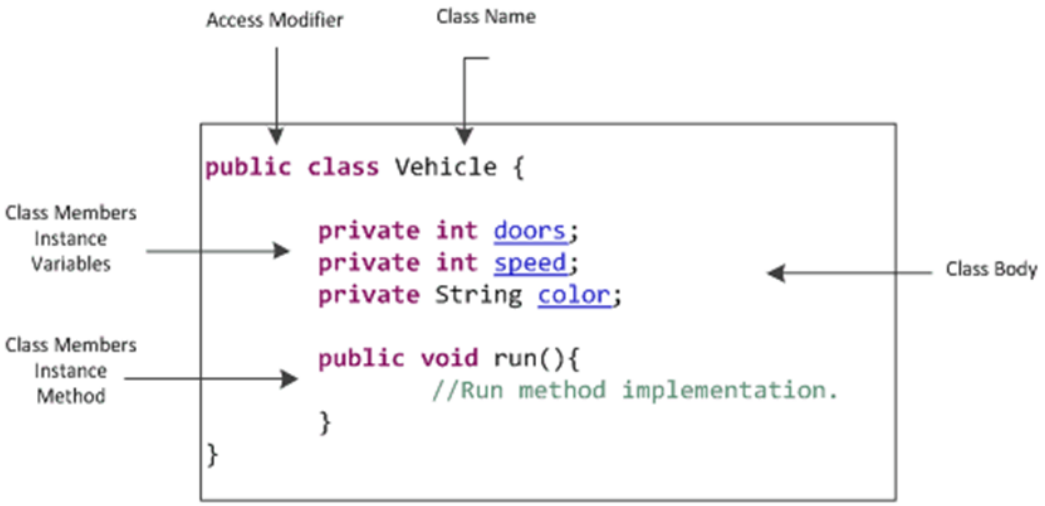
What Is a Class ?

A class is a blueprint or prototype from which objects are created

What Is an Object?

An object is a software bundle of related state and behavior.





Example of Class

```
class Vehicle
{
  private int door;
  private int speed;
  private string color;
  public void run()
        {
        System.out.println("Vehicle  is running speed is "+speed);
        }
}
```

Object Creation and Usage

```
Class VehicleUse
{
public static void main (String args[])
        {
                        Vehicle car=new Vehicle();
                        Vehicle bike=new Vehicle();
                        Vehicle raceCar=new Vehicle();

                        car.run();
                        Bike.run();
                        racecar.run();
        }
}
```

★ Java constructor

A constructor in Java is a special method that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes.

Note that the constructor name must match the class name, and it cannot have a return type (like void).

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

1. Default constructor (no-arg constructor)

```java
class Vehicle {
    // Creating a default constructor (not passed arguments)
    Vehicle() {
        System.out.println("Vehicle is created");
    }

    // Main method
    public static void main(String args[]) {
        // Calling a default constructor
        Vehicle b = new Vehicle();
    }
}
```

output

```
Vehicle is created
```

2. Parameterized constructor

```java
class Vehicle {
    String color; // Declare color as a class variable

    // Creating a constructor that accepts a color parameter
    Vehicle(String color) {
        this.color = color; // Assign the parameter value to the class variable
        System.out.println("Vehicle is created with color: " + color);
    }

    public static void main(String args[]) {
        // Calling the constructor by passing a color parameter
        Vehicle b = new Vehicle("red");
    }
}
```

output

```
Vehicle is created with color: red
```

★ Difference between constructor and method in Java

Constructor	Java method
A constructor is used to initialize the state of an object.	A method is used to expose the behavior of an object.
A constructor must not have a return type.	A method must have a return type.
The constructor is invoked implicitly.	The method is invoked explicitly.
The constructor name must be the same as the class name.	The method name may or may not be the same as the class name.
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.

★ Inheritance

A class that inherits from another class can reuse the methods and fields of that class.
Java, Inheritance means creating new classes based on existing ones.it is possible to inherit attributes and methods from one class to another. **Multiple inheritance is not supported in java.**

- superclass (parent / Base) - the class being inherited from
- subclass (child / Derived) - the class that inherits from another class

To inherit from a class, use the extends keyword.

Animal.java (Parent class)

```
class Animal {

   // field and method of the parent class
   String name;
   public void eat() {
        System.out.println("I can eat");
   }
}

class Main {
   public static void main(String[] args) {

        // create an object of the subclass
        Dog labrador = new Dog();

        // access field of superclass
        labrador.name = "Rohu";
        labrador.display();

        // call method of superclass
        // using object of subclass
        labrador.eat();

   }
}
```

answer

```
My name is Rohu
I can eat
```

Dog.java (Child class)

```java
// inherit from Animal
class Dog extends Animal {

    // new method in subclass
    public void display() {
        System.out.println("My name is " + name);
    }
}
```

Animal.java

```java
public class Animal {
    private boolean vegetarian;
    private String eats;
    private int noOfLegs;

    public Animal() {}

    public Animal(boolean veg, String food, int legs) {
        this.vegetarian = veg;
        this.eats = food;
        this.noOfLegs = legs;
    }

    public boolean isVegetarian() {
        return vegetarian;
    }

    public void setVegetarian(boolean vegetarian) {
        this.vegetarian = vegetarian;
    }

    public String getEats() {
        return eats;
    }

    public void setEats(String eats) {
        this.eats = eats;
    }
```

```
    public int getNoOfLegs() {
        return noOfLegs;
    }

    public void setNoOfLegs(int noOfLegs) {
        this.noOfLegs = noOfLegs;
    }

    public static void main(String[] args) {
        Cat cat = new Cat(false, "milk", 4, "black");
        System.out.println("Cat is Vegetarian? " + cat.isVegetarian());
        System.out.println("Cat eats " + cat.getEats());
        System.out.println("Cat has " + cat.getNoOfLegs() + " legs.");
        System.out.println("Cat color is " + cat.getColor());
    }
}
```

cat.java

```
public class Cat extends Animal {
    private String color;

    public Cat(boolean veg, String food, int legs) {
        super(veg, food, legs);
        this.color = "White";
    }

    public Cat(boolean veg, String food, int legs, String color) {
        super(veg, food, legs);
        this.color = color;
    }

    public String getColor() {
        return color;
    }

    public void setColor(String color) {
        this.color = color;
    }
}
```

Answer

```
Cat is Vegetarian? false
Cat eats milk
Cat has 4 legs.
Cat color is black
```

Advantages

- Code reusability
- Reduce duplicate code
- Reduce development time and effort
- Provide specialization
- Extendibility
  - Can extend an already made classes by adding new features

Disadvantage

- Tide coupling
- When the no of inheriting levels are increasing , maintenance is difficult

★ Encapsulation

The meaning of Encapsulation is to make sure that "sensitive" data is hidden from users.
To achieve this, you must:

declare class variables/attributes as private
provide public get and set methods to access and update the value of a private variable

Person.java **output , John**

```java
public class Person {
   private String name; // private = restricted
access

   // Getter
   public String getName() {
       return name;
   }

   // Setter
   public void setName(String newName) {
       this.name = newName;
   }
}
```

Main.java

```java
public class Main {
   public static void main(String[] args) {
       Person myObj = new Person();
       myObj.setName("John"); // Set the value of the name variable to "John"
       System.out.println(myObj.getName());
   }
}
```

Super Keyword in Java

The super keyword in Java is a reference variable which is used to refer to an immediate parent class object.

Whenever you create the instance of a subclass, an instance of the parent class is created implicitly which is referred to by a super reference variable.

Usage of Java super Keyword

- super can be used to refer to an immediate parent class instance variable.
- super can be used to invoke the immediate parent class method.
- super() can be used to invoke immediate parent class constructor

this Keyword in Java

this keyword refers to the current object in a method or constructor.

Usage of Java this keyword

- this can be used to refer to the current class instance variable.
- this can be used to invoke current class method (implicitly)
- this() can be used to invoke the current class constructor.
- this can be passed as an argument in the method call.
- this can be passed as an argument in the constructor call.
- this can be used to return the current class instance from the method.

★ Polymorphism

polymorphism is a feature that allows you to provide a single interface to varying entities of the same type.

Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.

Same name to different operations

- Overriding
- Overloading

1. Overriding

If a subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java.

Known as runtime polymorphism. A method declared final cannot be overridden. Static methods are not possible to override

Advantages

- Provide specific implementation for object
- Provide specific implementation for a method that is already provided by its super class.

Output

```
The animal makes a sound
The pig says: wee wee
The dog says: bow wow
```

```
class Animal {
   public void animalSound() {
       System.out.println("The animal makes a sound");
   }
}

class Pig extends Animal {
   public void animalSound() {
       System.out.println("The pig says: wee wee");
   }
}

class Dog extends Animal {
   public void animalSound() {
       System.out.println("The dog says: bow wow");
   }
}

class Main {
   public static void main(String[] args) {
       Animal myAnimal = new Animal();  // Create a Animal object
       Animal myPig = new Pig();  // Create a Pig object
       Animal myDog = new Dog();  // Create a Dog object
       myAnimal.animalSound();
       myPig.animalSound();
       myDog.animalSound();
   }
}
```

2. Overloading

With method overloading, multiple methods can have the same name with different parameters:

ex-

```
int myMethod(int x)
float myMethod(float x)
double myMethod(double x, double y
```

Answer

```
Total :30
Total :20
```

```java
class Calculation {
    int sum(int a, int b) {
        return a + b;
    }

    int sum(int a, int b, int c) {
        return a + b + c;
    }

    public static void main(String args[]) {
        Calculation obj = new Calculation();
        int tot;

        tot = obj.sum(10, 10, 10);
        System.out.println("Total :" + tot);

        tot = obj.sum(10, 10);
        System.out.println("Total :" + tot);
    }
}
```

Advantages of overloading

- Increases the readability
    - call a similar method for different types of data
- providing multiple behavior to same object with respect to attributes of object

Constructor overloading

A class can have two or more different implementations by having different constructors.

Output

```
Volume of mybox1 is 3000.0
Volume of mybox2 is 0.0
Volume of mycube is 343.0
```

‘

```java
class Box {
    double width, height, depth;

    // Constructor used when all dimensions specified
    Box(double w, double h, double d) {
        width = w;
        height = h;
        depth = d;
    }

    // Constructor used when no dimensions specified
    Box() {
        width = height = depth = 0;
    }

    // Constructor used when a cube is created
    Box(double len) {
        width = height = depth = len;
    }

    // Compute and return volume
    double volume() {
        return width * height * depth;
    }
}
```

Test.java

```java
public class Test {
    public static void main(String args[]) {
        // Create boxes using the various constructors
        Box mybox1 = new Box(10, 20, 15);
        Box mybox2 = new Box();
        Box mycube = new Box(7);

        double vol;

        // Get volume of the first box
        vol = mybox1.volume();
        System.out.println("Volume of mybox1 is " + vol);

        // Get volume of the second box
        vol = mybox2.volume();
        System.out.println("Volume of mybox2 is " + vol);

        // Get volume of the cube
        vol = mycube.volume();
        System.out.println("Volume of mycube is " + vol);
    }
}
```

★ Abstraction

Abstraction is the process of hiding certain details and only showing the essential features of the object.
Hiding the implementation details from the user
Abstraction can be achieved with either **abstract classes or interfaces**

Advantages

- Hiding implementation complexity from user
- Object is easy to used by users

The abstract keyword is a non-access modifier, used for classes and methods:

- **Abstract class -** is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).

- **Abstract method -** can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

```
// Abstract class
abstract class Animal {
   // Abstract method (does not have a body)
   public abstract void animalSound();
   // Regular method
   public void sleep() {
        System.out.println("Zzz");
   }
}

// Subclass (inherit from Animal)
class Pig extends Animal {
   public void animalSound() {
        // The body of animalSound() is provided here
        System.out.println("The pig says: wee wee");
   }
}

class Main {
   public static void main(String[] args) {
        Pig myPig = new Pig(); // Create a Pig object
        myPig.animalSound();
        myPig.sleep();
   }
}
```

```
The pig says: wee wee
Zzz
```

Interfaces

An interface is a completely "abstract class" that is used to group related methods with empty bodies:

```
// interface

interface Animal {

  public void animalSound(); // interface method (does not have a body)

  public void run(); // interface method (does not have a body)

}
```

To access the interface methods, the interface must be "implemented" (kinda like inherited) by another class with the implements keyword (instead of extends). The body of the interface method is provided by the "implement" class:

```
// Interface
interface Animal {
   public void animalSound(); // interface method (does not have a body)
   public void sleep(); // interface method (does not have a body)
}

// Pig "implements" the Animal interface
class Pig implements Animal {
   public void animalSound() {
        // The body of animalSound() is provided here
        System.out.println("The pig says: wee wee");
   }
   public void sleep() {
        // The body of sleep() is provided here
        System.out.println("Zzz");
   }
}

class Main {
   public static void main(String[] args) {
        Pig myPig = new Pig();  // Create a Pig object
        myPig.animalSound();
        myPig.sleep();
   }
}
```

```
The pig says: wee wee
Zzz
```

★ Exception Handling

Exception handling is the process of responding to unwanted or unexpected events when a computer program runs.

★ Program Errors

1. Syntax errors

Caused by an incorrectly written code such as misspelled keyword or variable, and incomplete code

2. Runtime errors

Occurs when a statement attempts an operation that is impossible to carry out
Ex: division by zero

3. Logical errors

Occurs when code compiles and runs without generating an error, but the result produced is incorrect

★ Types of Java Exceptions

1. **Checked exceptions**

IOException
SQLException
ClassNotFoundException
IllegalAccessException

- Occurs at the compile time
- Also called as compile time exceptions.
- Cannot be ignored at the time of compilation by programmers.
- All exceptions are checked exceptions

2. **Unchecked**

- Runtime exceptions

- At the time of execution.
- Never checked
- These include programming bugs, such as logic errors or improper use of an API.

ArithmeticException
NullPointerException
ArrayIndexOutOfBoundsException
InvalidArgumentException
NumberFormatException

- **Errors**

  - Not exceptions at all, but problems that arise beyond the control of the user or the programmer.
  - Errors are typically ignored in your code because you can rarely do anything about an error.

  - For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

OutOfMemoryError VirtualMachineError AssertionError

★ try catch block

The try statement allows you to define a block of code to be tested for errors while it is being executed.

The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

```
try
{
        //Protected code
}
Catch (ExceptionName e1)
{
        //Catch block
}
```

```
public class myclass
{
   public static void main(String[] args) {
        try{
            int a=4, b=0;
            int c=a/b;
            System.out.println(c);
        }
        catch(ArithmeticException E)
        {
            System.out.print("Error"+ E.toString());
        }
   }
}
```

```
catch ( Exception e ) {
    System.out.println ( "Something went wrong." );  - we can use this
  }
```

Output

```
Errorjava.lang.ArithmeticException: / by zero
Process finished with exit code 0
```

★ Multiple catch block

```
try
    {
                    //Protected code
    }
catch (ExceptionType1 e1)
    {
                   //Catch block
    }
catch (ExceptionType2 e2)
    {
                    //Catch block
    }
catch (ExceptionType3 e3)
    {
                   //Catch block
    }
```

★ finally Block

A finally block of code always executes, whether or not an exception has occurred.

```
public class Myclass {
   public static void main(String[] args) {
        try {
            int[] myNumbers = {1, 2, 3};
            System.out.println(myNumbers[10]);
        } catch (Exception e) {
            System.out.println("Something went wrong.");
        } finally {
            System.out.println("The 'try catch' is finished.");
        }
    }
}
```

```
Something went wrong.
The 'try catch' is finished.
```

★ Throwable

The Java throw keyword is used to explicitly throw an exception.Superclass of all errors and exceptions.The throw statement allows you to create a custom error.

The throw statement is used together with an exception type. There are many exception types available in Java: ArithmeticException, FileNotFoundException, ArrayIndexOutOfBoundsException, SecurityException, etc:

```
throw new ArithmeticException("not valid");
```

```
public class Myclass {
    static void checkAge(int age) {
        if (age < 18) {
                                                throw new
            ArithmeticException("Access denied - You must be at least       18
            years old.");
        }
        else {
            System.out.println("Access granted - You are old enough!");
        }
    }

    public static void main(String[] args) {
        checkAge(15); // Set age to 15 (which is below 18...)
    }
}
```

Output

```
Exception in thread "main" java.lang.ArithmeticException Create breakpoint : Access denied - You must be at least 18 years old.
    at Myclass.checkAge(Myclass.java:4)
    at Myclass.main(Myclass.java:12)
```

★ Nesting try blocks

```
public class Demo
{
   public static void main(String args[])
   {
        try
        {
            try
            {
                System.out.println("going to divide");
                int b =39/0;
            }
            catch(ArithmeticException e)
            {
                System.out.println(e);
            }
            System.out.println("other statement");
        }
        catch(Exception e){
            System.out.println("handeled");
        }
        System.out.println("normal flow..");
   }
}
```

Output

```
going to divide
java.lang.ArithmeticException: / by zero
other statement
normal flow..
```

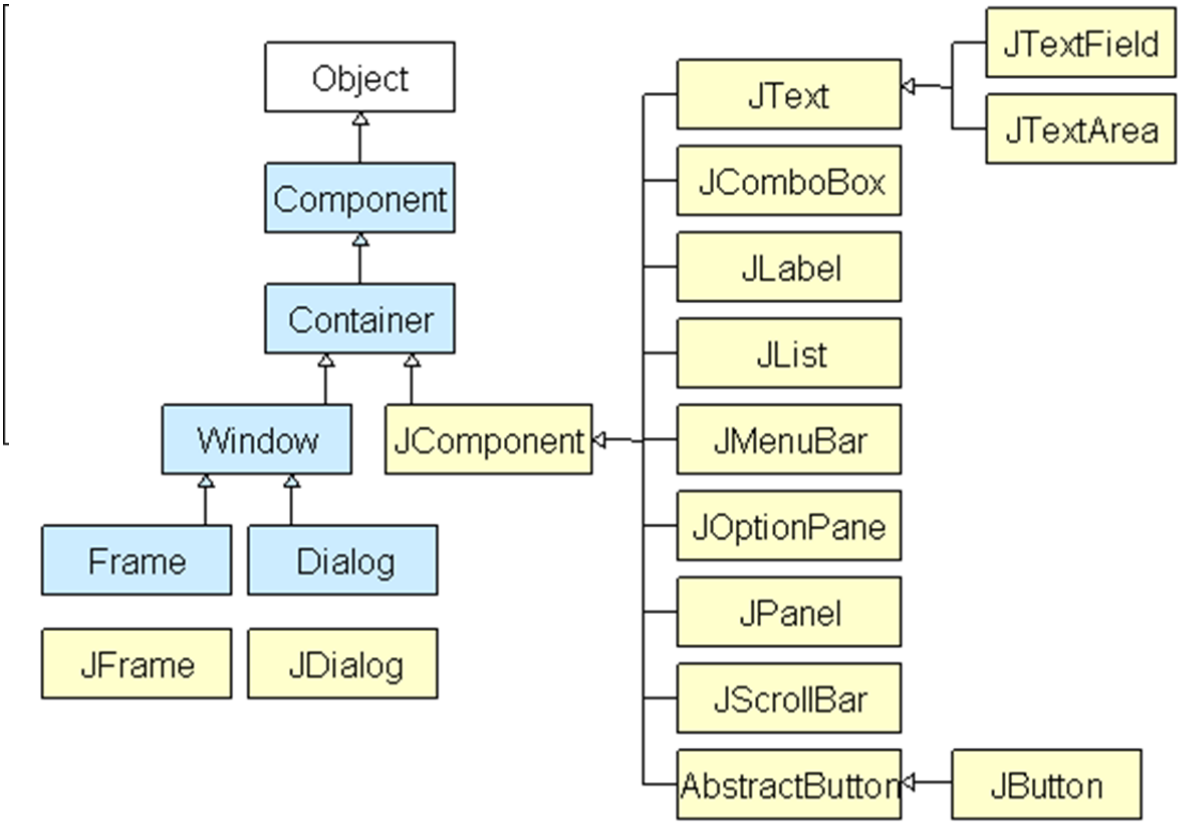
★ Swing

Swing API is set of extensible GUI Components to create JAVA Front End/ GUI Applications

Swing features

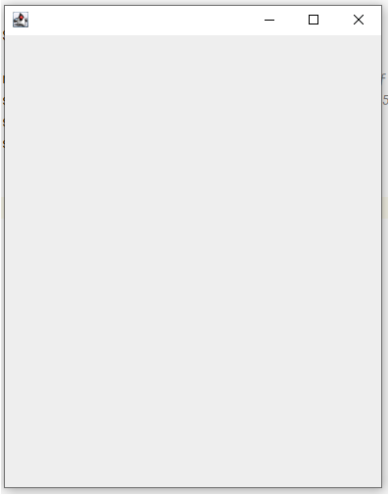
- Light Weight
- Rich controls
- Highly Customizable
- Pluggable look-and-feel- SWING

SWING - Controls

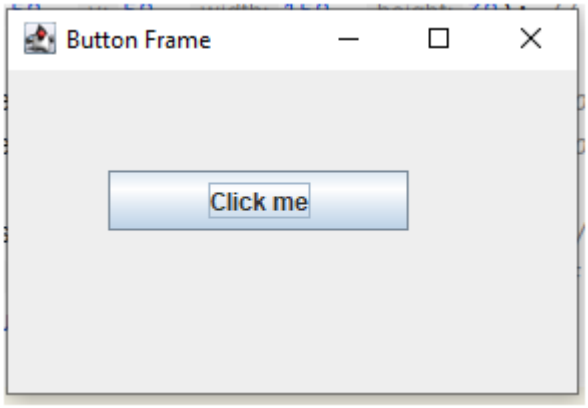


Frame

```
import javax.swing.*;

class TestSwing {
   public static void main(String[] args) {
        JFrame f = new JFrame(); // Creating an
instance of JFrame
        f.setSize(400, 500); // 400 width and 500
height
        f.setLayout(null); // Using no layout
managers
        f.setVisible(true); // Making the frame
visible
   }
}
```



```java
import javax.swing.*;

public class ButtonFrame {
   public static void main(String[] args) {
        JFrame frame = new JFrame("Button Frame"); // Create a JFrame with title "Button Frame"

        JButton button = new JButton("Click me"); // Create a JButton with label "Click me"
        button.setBounds(50, 50, 150, 30); // Set the button's position (x, y) and size (w,h)

        frame.getContentPane().setLayout(null); // Use null layout manager
        frame.getContentPane().add(button); // Add the button to the content pane of the frame

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Set default close operation
        frame.setSize(300, 200); // Set the size of the frame
        frame.setVisible(true); // Make the frame visible
   }
}
```

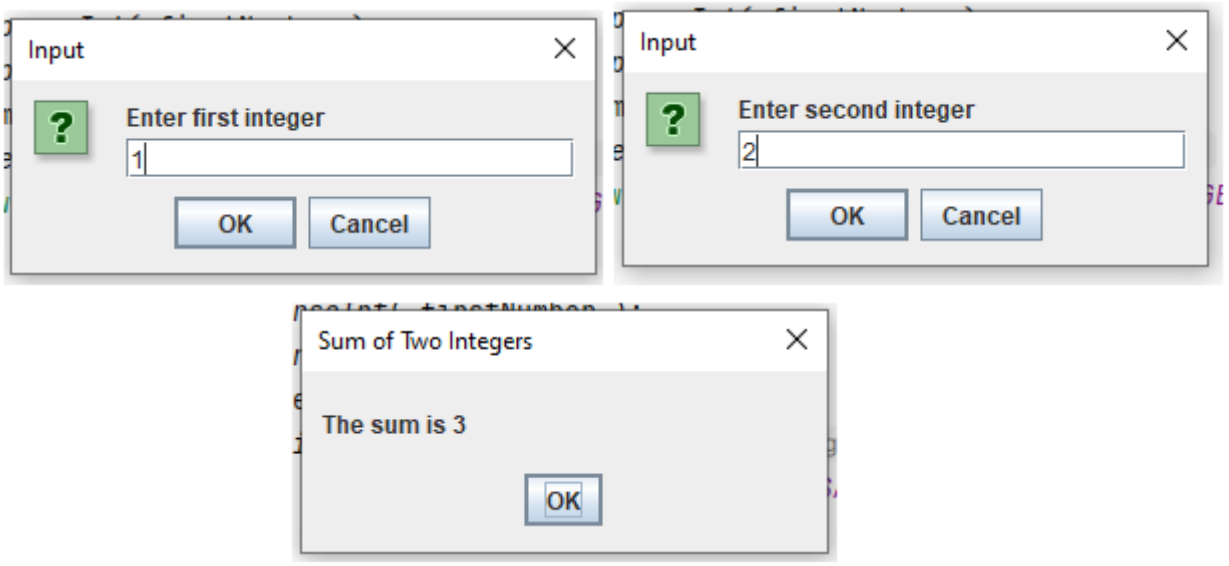


JOptionPane

- Display a message (through the use of the **showMessageDialog** method)
- Ask for user's confirmation (using the **showConfirmDialog** method)
- Obtain the user's input (using the **showInputDialog** method)
- Do the combined three above (using the **showOptionDialog** method)

```java
import javax.swing.JOptionPane;
public class Addition
{
   public static void main( String args[ ] )
   {
        String firstNumber =
                JOptionPane.showInputDialog( "Enter first integer" );
        String secondNumber =
                JOptionPane.showInputDialog( "Enter second integer" );
        int number1 = Integer.parseInt( firstNumber );
        int number2 = Integer.parseInt( secondNumber );
        int sum = number1 + number2; // add numbers
        JOptionPane.showMessageDialog( null, "The sum is " + sum,
                "Sum of Two Integers", JOptionPane.PLAIN_MESSAGE );
   }
}
```

- **JOptionPane.PLAIN_MESSAGE**- This type simply displays the message in a dialog box without any specific icon.
- **JOptionPane.INFORMATION_MESSAGE** - Displays an information icon along with the message.
- **JOptionPane.WARNING_MESSAGE** - Shows a warning icon along with the message.
- **JOptionPane.ERROR_MESSAGE -** Displays an error icon along with the message.
- **JOptionPane.QUESTION_MESSAGE -** Shows a question mark icon along with the message.



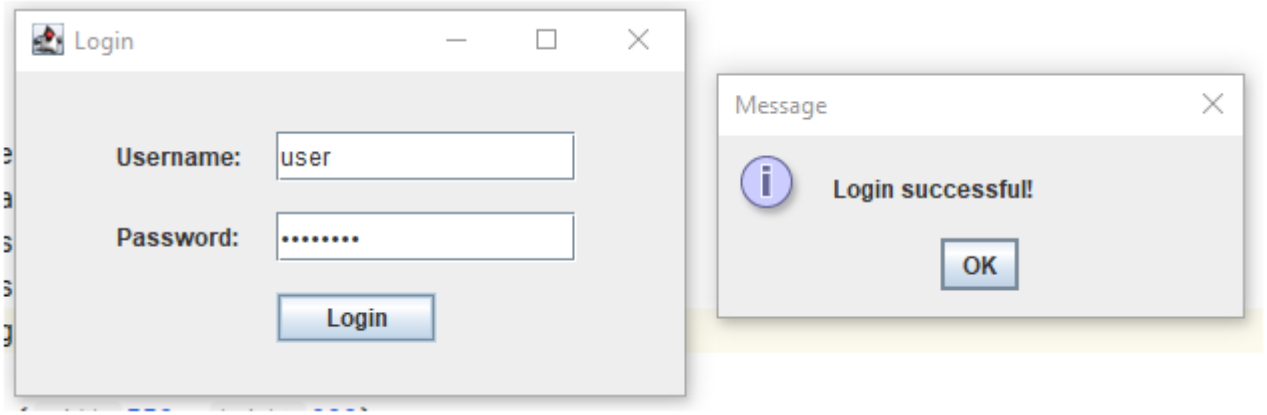
★ Event

java.awt.event

Package defines classes and interfaces used for event handling in the AWT and Swing.

```
import javax.swing.*;
import java.awt.event.*;

public class LoginFrame {
   public static void main(String[] args) {
        JFrame frame = new JFrame("Login");

        JLabel userName = new JLabel("Username:");
        userName.setBounds(50, 30, 80, 25);
        JTextField usanameInput = new JTextField();
        usanameInput.setBounds(130, 30, 150, 25);

        JLabel password = new JLabel("Password:");
        password.setBounds(50, 70, 80, 25);
        JPasswordField passwordInput = new JPasswordField();
        passwordInput.setBounds(130, 70, 150, 25);

        JButton loginButton = new JButton("Login");
        loginButton.setBounds(130, 110, 80, 25);
```

```
        // Action Listener for the login button
        loginButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String username = usanameInput.getText();
                String password = String.valueOf(passwordInput.getPassword());

                //Perform login validation (Replace with actual validation logic)
                if (username.equals("user") && password.equals("password")) {
                    JOptionPane.showMessageDialog(frame, "Login successful!");
                } else {
                   JOptionPane.showMessageDialog(frame, "Invalid username or
                       password.", "Login Error", JOptionPane.ERROR_MESSAGE);
                }
            }
        });

        frame.add(userName);
        frame.add(usanameInput);
        frame.add(password);
        frame.add(passwordInput);
        frame.add(loginButton);

        frame.setSize(350, 200);
        frame.setLayout(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```



★ Difference between AWT and Swing

Java AWT	Java Swing
AWT components are platform-dependent.	Java swing components are platform-independent.
AWT components are heavyweight.	Swing components are lightweight.
AWT doesn't support pluggable look and feel.	Swing supports pluggable look and feel.

AWT provides less components than Swing	Swing provides more powerful components such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
AWT doesn't follows MVC (Model View Controller)	Swing follows MVC.

★ Stream

Java uses the concept of stream to make I/O operation fast.A stream is a sequence of data

- System.out: standard output stream

  System.out.println("simple message");

- System.in: standard input stream

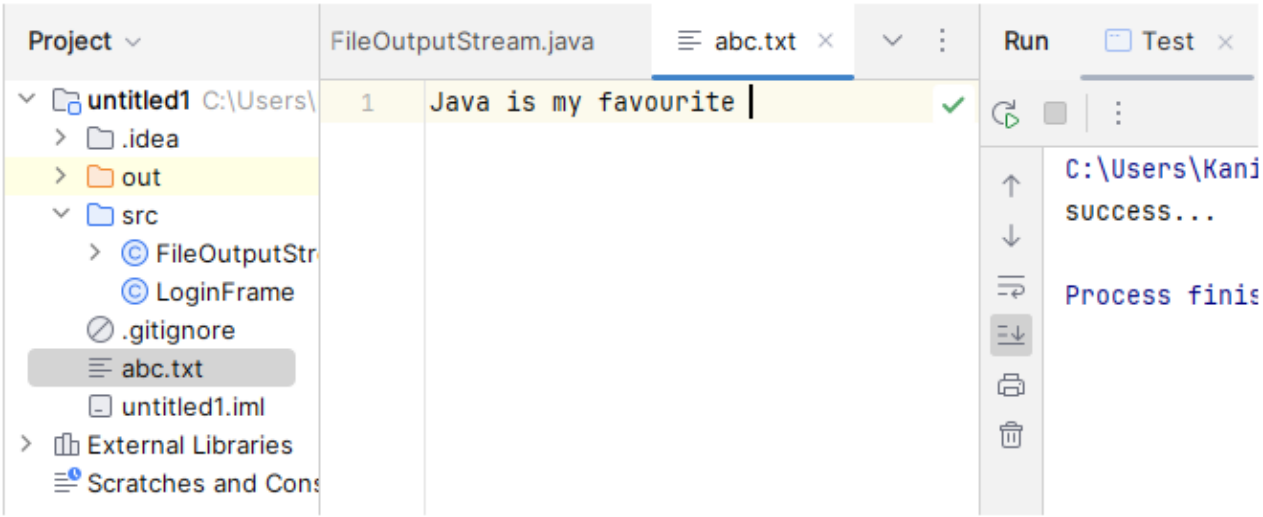
  int i=System.in.read();
  //returns ASCII code of 1st character

- System.err: standard error stream

  System.err.println("error message");

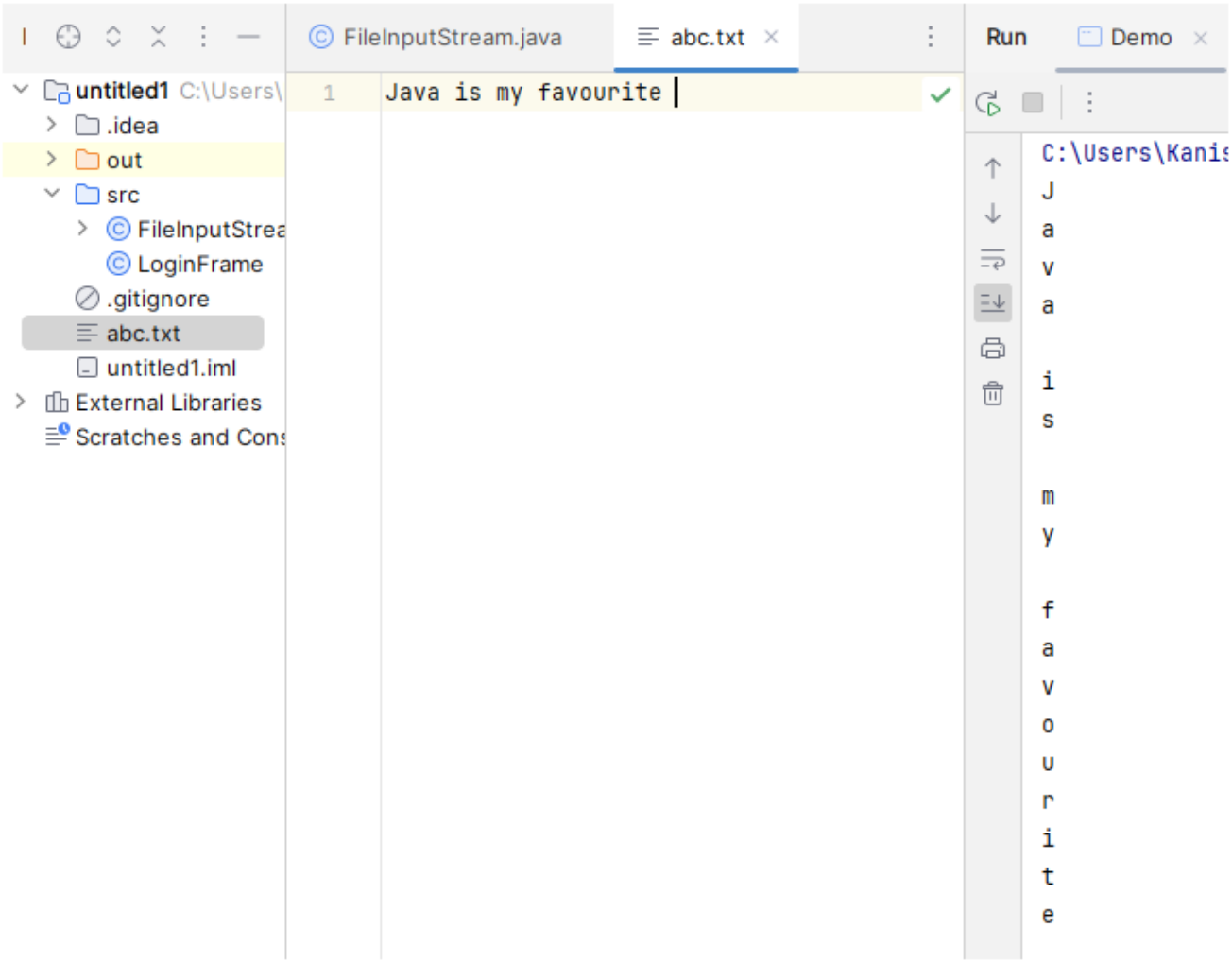
FileOutputStream class

```
import java.io.*;
class Test{
   public static void main(String args[])
   {
        try
        {
            FileOutputStream fout=new FileOutputStream("abc.txt");
            String s="Java is my favourite ";
            byte b[]=s.getBytes();//converting string into byte array
            fout.write(b);
            fout.close();
            System.out.println("success...");
        }
        catch(Exception e){System.out.println(e);}
   }
}
```



FileInputStream class

```java
import java.io.*;

class Demo {
   public static void main(String args[]) {
       try {
           // Creating a FileInputStream to read the file "abc.txt"
           FileInputStream fin = new FileInputStream("abc.txt");

           int i = 0;
           // Reading the file byte by byte until the end of the file
           while ((i = fin.read()) != -1) {
               // Casting the byte value to char and printing it to the console
               System.out.println((char) i);
           }

           // Closing the FileInputStream
           fin.close();
       } catch (Exception e) {
           // Handling any exceptions that might occur during file operations
           System.out.println(e);
       }
   }
}
```



★ JDBC (Java Database Connectivity)

JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database.

★ What is API ?

API (Application programming interface) is a document that contains a description of all the features of a product or software.

★ Java Database Connectivity

There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:

1. Register the Driver class
2. Create connection
3. Create statement
4. Execute queries
5. Close connection

Java Database Connectivity with MySQL

Driver class: The driver class for the mysql database is **com.mysql.jdbc.Driver.**

Connection URL:

The connection URL for the mysql database is **jdbc:mysql://localhost:3306/sonoo** where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and sonoo is the database name. We may use any database, in such case, we need to replace the sonoo with our database name.

Username:

The default username for the mysql database is root.

Password:

It is the password given by the user at the time of installing the mysql database

first create a table in the mysql database

create database stu;
create table student (id int,name varchar(10),marks int);

```
import java.sql.*;

class MysqlCon {
   public static void main(String args[]) {
       try {
           Class.forName("com.mysql.jdbc.Driver");
           Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306
            /stu", "root", " ");

           // here stu is database name, root is username and password is empty
           Statement stmt = con.createStatement();
           ResultSet rs = stmt.executeQuery("select * from student");
           while (rs.next())
           System.out.println(rs.getInt(1) + "  " + rs.getString(2) + "  " +
           rs.getString(3));
           con.close();
       } catch (Exception e) {
           System.out.println(e);
       }
   }
}
```