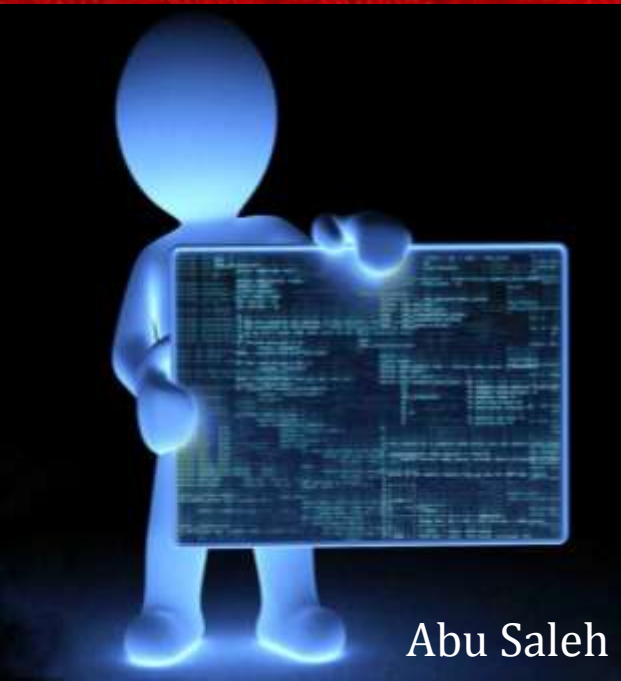


A Guided Book of

Object Oriented Programming with JAVA



Abu Saleh Musa Miah(Abid)

A Guidebook of JAVA

Engr. Abu Saleh Musa Miah (Abid)
Lecturer, Rajshahi Engineering Science and Technology College(RESTC)
B.Sc.Engg. in Computer Science and Engineering(First Class First)
M.Sc.Engg.(CSE-Pursuing),University of Rajshahi.
Email:abusalehcse.ru@gmail.com
Cell:+88-01734264899

Object Oriented Programming with CPP

=====	
=====	
Writer	: Engr. Abu Saleh Musa Miah (Abid). B.Sc.Engg. in Computer Science and Engineering(First Class First) M.Sc.Engg.(CSE-Pursuing),University of Rajshahi. Cell:+88-01734264899
Email	: abusalehcse.ru@gmail.com
Publisher	: Abu Syed Md Mominul Karim Masum. Chemical Engr. Chief Executive Officer (CEO) Swarm Fashion, Bangladesh. Mohakhali,Dhaka.
Email	: swarm.fashion@gmail.com
Cover page Design	: Md. Kaiyum Nuri Chief Executive Officer (CEO) Jia Shah Rich Group(Textile) MBA,Uttara University
First Publication	: Octobor-2016
Copyright	: Writer
Computer Compose	: Writer
Print	: Royal Engineering Press & Publications. Meherchandi, Padma Residential Area, Boalia, Rajshahi.
Reviewer Team	: Engr. Syed Mir Talha Zobaed. B.Sc. Engg. (First Class First) M.Sc. Engineering (CSE) University of Rajshahi Omar Faruk Khan (Sabbir) M. Engineering (CSE) University of Rajshahi
PRICE	: 400.00 TAKA (Fixed Price). US 05 Dollars (Fixed Price).

N.B: All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, Electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher. It is hereby declared that, if someone or somebody copy the book or try to copy the book as partially or wholly for personal use or merchandizing, is punishable offence and the Publisher may take lawful action and can demand 10, 00, 000 (Ten lacks) TK as compensation against them.

OOP CPP	: Engr. Abu Saleh Musa Miah (Abid).
Published by	: Royal Engineering Press, Bangladesh Meherchandi, Padma R/A, Boalia, Rajshahi.

DEDICATED

TO

Rangpur Engineering College(REC)

[This Page is Left Blank Intentionally]

লেখকের কথা

Object Oriented Programming with JAVA কোর্সটি বাংলাদেশের প্রায় সকল বিশ্ববিদ্যালয়ের সিলেবাসে স্থান লাভ করেছে, বাজারে টেক্সটবই থাকলেও গভীর ভাবে অনুধাবন ও পরীক্ষায় ভাল মার্কস উঠানোর মত সাজানো টেক্সটবই সহজলভ্য নয়। পরীক্ষার প্রস্তুতির স্বার্থে তাই, এই গাইডবইটি আপনাকে সহযোগিতা করবে বলে আশা করা যায়।

এই বইয়ের পাঠক হিসেবে, আপনিই হচ্ছেন সবচেয়ে গুরুত্বপূর্ণ সমালোচক বা মন্তব্যকারী। আর আপনাদের মন্তব্য আমার কাছে মূল্যবান, কারণ আপনি বলতে পারবেন আপনার উপযোগী করে বইটি লেখা হলো কিনা অর্থাৎ বইটি কিভাবে প্রকাশিত হলে আরও ভাল হতো। সামগ্রিক ব্যাপারে আপনাদের যে কোন পরামর্শ আমাকে উৎসাহিত করবে।

Engr. Abu Saleh Musa Miah (Abid)
Writer

ACKNOWLEDGEMENTS

I wish to express my profound gratitude to all those who helped in making this book a reality; especially to my families, the classmates, and authority of Royal Engineering Publications for their constant motivation and selfless support. Much needed moral support and encouragement was provided on numerous occasions by my families and relatives. I will always be grateful, to the numerous web resources, anonymous tutorials hand-outs and books I used for the resources and concepts, which helped me build the foundation.

I would also like to express my profound gratitude to Engr. Syed Mir Talha Zobaed for his outstanding contribution in inspiring, editing and proofreading of this guidebook. I am also grateful to Omar Faruque Khan (Sabbir) for his relentless support and guideline in making this book a reality.

I am thankful to the following readers those have invested their valuable times to read this book carefully and have given suggestion to improve this book:

Dr. Nirod Boron Nath(Principle,REC)

Md. Murad Ali (Lecturer, REC).

Md. Mahmudul Hasan (Lecturer, ICE Department REC)

..... And numerous anonymous readers.

I am also thankful to the different hand notes from where I have used lots of solutions, such as Dynamic Memory Allocation, Operator overloading etc

I wish to express my profound gratitude to the following writers whose books I have used in my Guidebooks:

1	Herbert Schildt	Java: The Complete Reference Third Edition
2	Steve Ouallin	Practical Programming

..... and Numerous anonymous Power Point Slides and PDF chapters from different North American Universities.



Object Oriented Programming with JAVA Syllabus

CSE1221: Object Oriented Programming with JAVA
75 Marks [70% Exam, 20% Quizzes/Class Tests, 10% Attendance]
3 Credits, 33 Contact hours, Exam. Time: 4 hours

Section-A:

Concepts of Object Oriented Programming: Class, Object, Abstraction, Encapsulation, Inheritance, Polymorphism.

Introduction to Java: History of Java, Java features and advantages, creating classes with Java, Concept of Constructors, Using JDK, Java application and Applet, Variables, Data types, Arrays, Operators and control flow.

Methods: Using methods, declaring a class method, Implementation of inheritance, calling a class method, passing Parameters, Local variables and variable scope.

Using Standard Java Packages: Creating graphical user interfaces with AWT, Managing graphics objects with GUI layout managers, Event handling of various components.

Exception Handling: Overview of exception handling, the basic model, Hierarchy of event classes, Throw clause, Throws statement, try-catch block.

Section-B:

Streams and Input/output Programming: Java's file management techniques, Stream manipulation classes.

Thread: Thread, Multithread, Synchronization, Deadlock, Thread scheduling.

Socket Programming: Socket basics, Socket-based network concepts, Client server basics, Client server algorithm, Socket for client, Socket for server.

Java Database Connectivity: JDBC, JDBC drivers, the JAVA.sql packages, SQL, JDBC connection and executing SQL, The process of building a JAVA application.

Advanced Java Programming: Java Servlets and Servlets architectures, RMI, Multimedia, Java Beans, Java server Pages.

Books Recommended:

1. H. Schidt : **C++: A Beginner's Guide, McGraw Hill**
1. H. Schidt : **C++: The Complete Reference, McGraw Hill**
2. N. Barkakati : **Object Oriented Programming with C++, Prentice Hall India**
3. B. Stroustrap : **The C++ Pr**

Index

Part – II: CPP

CHAPTER 1 Concepts of Object Oriented Programming

CHAPTER 2 Introduction to Java

CHAPTER 3 Methods:

CHAPTER 4 Using Standard Java Packages

CHAPTER 5 Exception Handling

CHAPTER 6 Streams and Input/output Programming

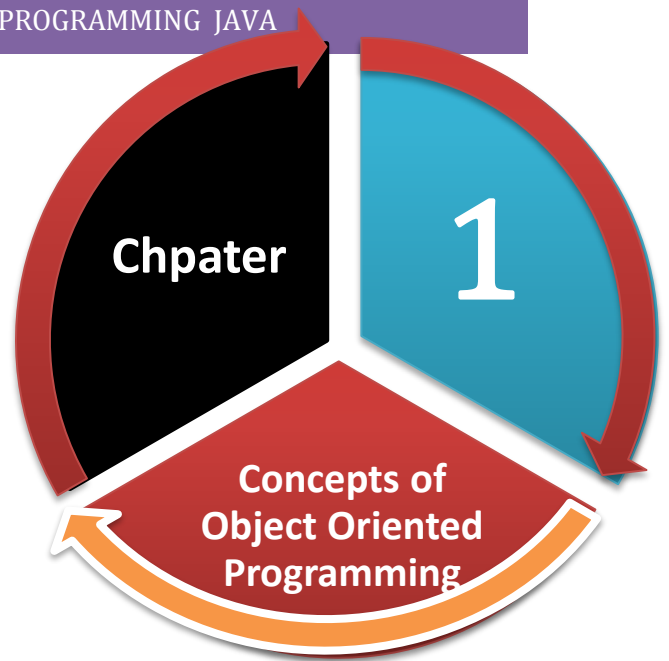
CHAPTER 7 Thread

CHAPTER 8 Socket Programming

CHAPTER 9 Java Database Connectivity

CHAPTER 10 Advanced Java Programming

Important Question:



1. What is byte code? **Exam-2015**
2. Write down the features of java. Explain any two of them. **Exam-2015**
3. How does java implement polymorphism? Write a Java program that describes polymorphism. **Exam-2015**
4. Define the terms i)inheritance ii)Encapsulation. Exam-2014
5. Differentiate between source code and byte code. Exam-2014
6. What is object oriented language? Briefly discuss about the term Polymorphism? Exam-2013
7. What is class ?Give an example of class. Write down the general form of a class definition. Exam-2013
8. What are the major C++ features that were intentionally omitted from Java or significantly modified? Exam-2013

Question:What is byte code? Exam-2015

Answer:

Bytecode is computer object code that is processed by a program, usually referred to as a virtual machine, rather than by the "real" computer machine, the hardware processor.

The virtual machine converts each generalized machine instruction into a specific machine instruction or instructions that this computer's processor will understand. Bytecode is the result of compiling source code written in a language that supports this approach. Most computer languages, such as C and C++, require a separate compiler for each computer platform- that is, for each computer operating system and the hardware set of instructions that it is built on. Windows and the Intel line of microprocessor architectures are one platform; Apple and the PowerPC processors are another. Using a language that comes with a virtual machine for each platform, your source language statements need to be compiled only once and will then run on any platform.

Question: What is object oriented language? Briefly discuss about the term Polymorphism? Exam-2013

Answer:

Object Oriented Programming:

A type of programming in which programmers define not only the data type of a data structure, but also the types of operations (functions or methods) that can be applied to the data structure. In this way, the data structure becomes an *object* that includes both data and functions or method

- 📖 OOP allows us to decompose a problem into number of entities called objects and then build data and methods (functions) around these entities.
- 📖 The data of an object can be accessed only by the methods associated with the object.
- 📖 An object-oriented programming language provide support for the following object oriented concepts
 - Class
 - Object
 - Abstraction
 - Encapsulation
 - Inheritance
 - Polymorphism

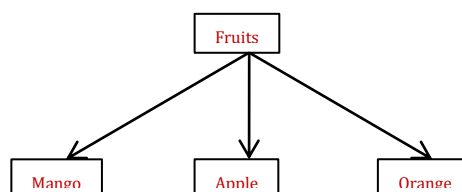
Question: What is class? Give an example of class. Write down the general form of a class definition. Exam-2013

Class:

A class can be defined as a template/blueprint that describes the behavior/state that the object of its type support

- ❖ A *class* is simply a representation of a type of *object*
- ❖ For example, mango, apple, orange are the members of the class fruits
- ❖ If fruit has been defined as a class, then the statement

Fruits mango;



- ❖ It will create an object mango belonging to the class fruit
- ❖ A class is created by using the keyword **class**

The general form of a class definition is

```
class classname {  
    // declare instance variables  
    type var1;  
    type var2;  
    ...  
    type varN;  
    // declare methods  
    type method1(parameters)  
    { // body of method }  
    type method2(parameters)  
    { // body of method }  
    // ...  
    type methodN(parameters)  
    { // body of method }  
}
```

Example Program

```
public class HelloWorld  
{  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}  
  
public class Sample {  
    public static void main(String[] args) {  
        int a = 30;  
        int b = 45;  
        Addi(a, b);  
        Sub(a,b);  
    }  
    public static void Addi(int a, int b) {  
        int c = a+b;  
        System.out.println("Addition = " + c);  
    }  
  
    public static void Sub(int a, int b) {  
        int d = a+b;  
        System.out.println("Sub = " + d);  
    }  
}
```

Object:

An Object is a software entity that models something in the real world. It has two main

Properties:

- **State:** the object encapsulates information about itself - attributes or fields.
- **Behavior:** the object can do some things on behalf of other objects – methods
Vehicle minivan = new Vehicle ();

📖 The above line is used to declare an object of type **Vehicle**

📖 This declaration performs two functions

- It declares a variable called minivan of the class type Vehicle. This variable does not define an object. Instead, it is simply a variable that can refer to an object
- The declaration creates a physical copy of the object and assigns to minivan a reference to that object

📖 This is done by using the **new** operator

📖 Thus, in Java, all class objects must be dynamically allocated

- Vehicle minivan; // declare reference to object
- minivan = new Vehicle(); // allocate a Vehicle object

Question: Write the different between class and object?

Answer:

Class	Object
A description of the <i>common properties of a</i> set of objects	A representation of the <i>properties of a single</i> instance
A concept	A phenomenon
A class is a part of a program	An object is part of data and a program execution
Example 1: Person	Example 1: Bill Clinton, Bono, Viggo Jensen

A phenomenon is a thing in the “real” world that has individual existence

A concept is a generalization, derived from a set of phenomena and based on the common properties of these phenomena

Question: Write down the features of java. Explain any two of them. Exam-2015

Question: Define the terms i) inheritance ii) Encapsulation. Exam-2014

Question: How does java implement polymorphism? Write a Java program that describes polymorphism.Exam-2015

Question: What are the major C++ features that were intentionally omitted from Java or significantly modified? Exam-2013

Feature of JAVA/OOP:

- 📖 Abstraction
- 📖 Encapsulation
- 📖 Polymorphism
- 📖 Enheritance

Abstraction:

Abstraction refers to the act of representing essential features without including the background details or explanations. Classes use the concept of abstraction and are defined as a list of abstract attributes. Abstraction is the representation of the essential features of an object. Abstraction means ignoring irrelevant features, properties, or functions and emphasizing the relevant ones. Abstraction is the concept of describing something in simpler terms i.e abstracting away the details, in order to focus on what is important

Encapsulation

Encapsulation is the mechanism that binds together code and the data it manipulates, and keeps both safe from outside interference and misuse.

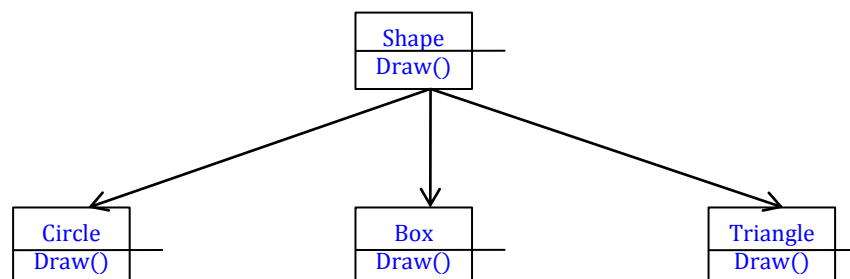
- 📖 The technical term for combining data and functions together as a bundle is encapsulation.
- 📖 Storing data and functions in a single unit (class) is encapsulation. Data cannot be accessible to the outside world and only those functions which are stored in the class can access it.
- 📖 Encapsulation is a technique. It may or may not be for aiding in abstraction, but it is certainly about data hiding
- 📖 The insulation from direct access by the program is called data hiding

Polymorphism:

Polymorphism is the attribute that allows one interface to control access to a general class of actions. The specific action selected is determined by the exact nature of the situation.

The behavior depends upon the type of data

- Ex: The operation of addition (Integer and string)



- 📖 Polymorphism play an important role in allowing objects having different internal structures to share the same external interface
- 📖 This means that a general class of operations may be accessed in the same manner even though specific actions associated with each operation may differ

Inheritance

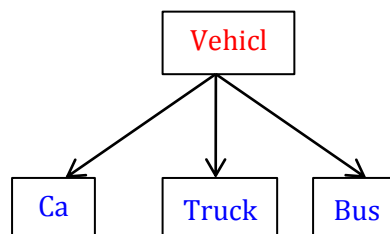
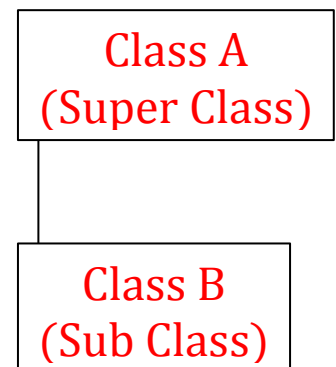
Inheritance is the process by which one object can acquire the properties of another object. This is important because it supports the concept of classification.

- 📖 Inheritance is the process by which objects of one class acquire the properties of objects of another class
- 📖 This is important because Inheritance support the concept of hierarchical classification
- 📖 In OOP, the concept of heritance provides the idea of reusability
- 📖 This means that we can add additional features to an existing class without modifying it
- 📖 This is possible by deriving a new class from the exiting one
- 📖 The new class will have the combined features of both the classes

class B extends A

```
{  
.  
././additionsto,andmodifications  
of,  
././stuffinheritedfromclassA
```

- Inheritance (Example):



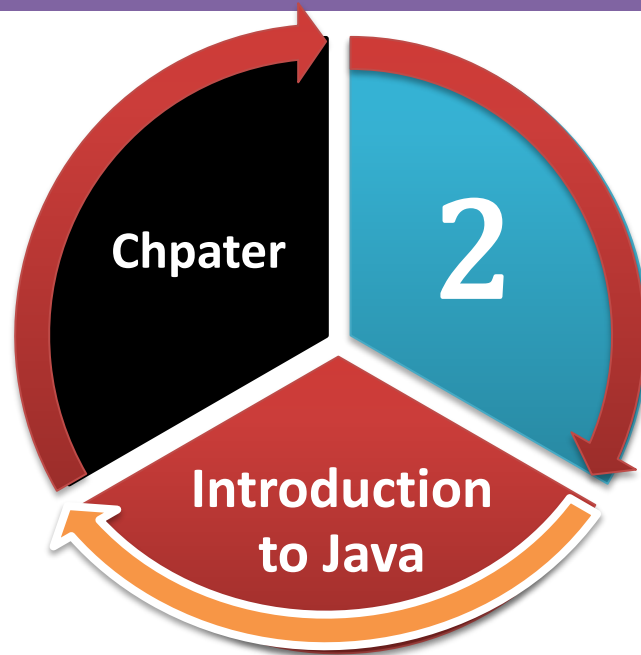
```
class Vehicle {  
    int registrationNumber;  
    Person owner; // ( Assuming t h a t a Person class has been def  
    ined ! )  
    void transferOwnership(Person newOwner) {  
        ... .....}  
    ...  
}  
    class Car extends Vehicle {  
        int numberOfDoors;  
        ...  
    }  
    class Truck extends Vehicle {  
        int numberOfAxels;  
        ...  
    }  
    class Bus extends Vehicle {  
        boolean hasSidecar;  
        ...  
    }  
}
```

Question: Write the Advantage of Java?

Advantage of Java:

- ▶ Java Is Simple
- ▶ Java Is Object-Oriented

- ▶ Java Is Distributed
- ▶ Java Is Interpreted
- ▶ Java Is Robust
- ▶ Java Is Secure
- ▶ Java Is Architecture-Neutral
- ▶ Java Is Portable
- ▶ Java's Performance
- ▶ Java Is Multithreaded
- ▶ Java Is Dynamic



Important Question

1. What is Java ?Briefly describe the following features of Java programming language;
Exam-2014
2. Distinguish between JDK and JRE. **Exam-2013**
3. Platform independent and Portable .ii) Robust and secure. Exam-2014
4. What is an array? Write a Java program that can print the value 0 to10 by using single dimension array. Exam-2014
5. Write down the name of different primitive data types in java? **Exam-2015**
6. Explain th structure of java program. **Exam-2015**
7. What is a static member variable? Using the concept of static variable, write a Java program that shows the number of objects created from a class. Exam-2013
8. What will be the output of the following program? Identify and correct the errors of the following program if necessary? **Exam-2015**

Question: What is Java? Briefly describe the following features of Java programming language; Exam-2014

Answer:

Java:

Java is a general-purpose computer programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible.

Question: Define i) Platform independent and Portable .ii) Robust and secure. Exam-2014

Answer:

Platform independent:

Platform independence means that the same program works on any **platform** (operating system) without needing any modification. In the case of **Java** the application runs in a **Java Virtual Machine** which itself isn't **platform independent**.

Portable:

. "**Java is portable**" refers to the SE version. It means that we can run **Java** bytecode on any hardware that has a compliant JVM. It doesn't mean that ME is the same as SE is the same as EE.

Portability is a measure for the amount of effort to make a program run on another environment than where it originated.

Java goes further than just being architecture-neutral:

- 📖 No "implementation dependent" notes in the spec (arithmetic and evaluation order)
- 📖 Standard libraries hide system differences
- 📖 The java environment itself is also portable: the portability boundary is posix compliant

Robust:

Robust means reliable and no programming language can really assure reliability. Java puts a lot of emphasis on early checking for possible errors, as Java compilers are able to detect many problems that would first show up during execution time in other languages.

Java has been designed for writing highly reliable or robust software:

- 📖 Language restrictions (e.g. no pointer arithmetic and real arrays) to make it impossible for applications to smash memory (e.g. overwriting memory and corrupting data)
- 📖 Java does **automatic garbage collection**, which prevents memory leaks
- 📖 Extensive compile-time checking so bugs can be found early; this is repeated at runtime for flexibility and to check consistency

Java has the strong memory allocation and automatic garbage collection mechanism. It provides the powerful exception handling and type checking mechanism as compare to other programming languages. Compiler checks the program whether there any error and interpreter checks any run time error and makes the system secure from crash. All of the above features make the java language robust.

Java example of "robust" code:



```
if (var == true) {  
    ...  
} else if (var == false) {
```

```
    ...  
  } else {  
    ...  
  }
```

"robust code" means that our program takes into account all possibilities, and that there is no such thing as an error - all situations are handled by the code and result in valid state, hence the "else".

Secure:

Security is an important concern, since Java is meant to be used in networked environments. Without some assurance of security, you certainly wouldn't want to download an applet from a random site on the net and let it run on your computer. Java's memory allocation model is one of its main defenses against malicious code (e.g can't cast integers to pointers, so can't forge access). Furthermore:

-  access restrictions are enforced (public, private)
-  byte codes are verified, which copes with the threat of a hostile compiler

Although in the case of Secure, it often refers to

- 1) strongly typed
- 2) No pointer access
- 3) Byte code verifier

Question: Briefly describe the history of Java?

Answer:

History of Java:

Java evolved from C++, which evolved from C, which evolved from BCPL and B

- ▶ Microprocessors are having a profound impact in intelligent consumer-electronic devices.
- ▶ 1991
 - Recognizing this, Sun Microsystems funded an internal corporate research project, which resulted in a C++-based language named Java
 - Created by James Gosling.
- ▶ 1993
 - The web exploded in popularity
 - Sun saw the potential of using Java to add dynamic content to web pages.
- ▶ Java garnered the attention of the business community because of the phenomenal interest in the web.

Question: Distinguish between JDK and JRE. Exam-2013**Answer:**

JDK includes the **JRE** plus command-line development tools such as compilers and debuggers that are necessary or useful for developing applets and applications.

The **difference between JDK and JRE:**

JDK	JRE
The JDK is a superset of the JRE, and contains everything that is in the JRE, plus tools such as the compilers and debuggers necessary for developing applets and applications	The Java Runtime Environment (JRE) provides the libraries, the Java Virtual Machine, and other components to run applets and applications written in the Java programming language
JDK is the software development kit for java	while JRE is the place where you run your programs
It's the full featured Software Development Kit for Java, including JRE, and the compilers and tools (like JavaDoc, and Java Debugger) to create and compile programs.	Java Runtime Environment. It is basically the Java Virtual Machine where your Java programs run on. It also includes browser plugins for Applet execution.
if we are planning to do some Java programming, you will also need JDK .	when we only care about running Java programs on your browser or computer you will only install JRE
Sometimes, even though you are not planning to do any Java Development on a computer, you still need the JDK installed	The JRE is, as the name implies, an <i>environment</i> . It's basically a bunch of directories with Java-related files.
The JDK is also a set of directories	It looks a lot like the JRE but it <i>contains</i> a directory (called JRE) with a complete JRE, and it has a number of development tools, most importantly the Java compiler <code>javac</code> in its <code>bin</code> directory.

. Question: what is JVM?**JVM:**

The **Java Virtual machine (JVM)** is the virtual machine that runs the Java bytecodes. The JVM doesn't understand Java source code, that's why you compile your *.java files to obtain *.class files that contain the bytecodes understood by the JVM. It's also the entity that allows Java to be a "portable language" (write once, run anywhere). Indeed, there are specific implementations of the JVM for different systems (Windows, Linux, MacOS, see the Wikipedia list), the aim is that with the same bytecodes they all give the same results.

**Question: What are the difference between CPP and Java programming language?
Discuss with example?**

Difference between C++ and Java

C++	Java
Compatible with C source code, except for a few corner cases.	No backward compatibility with any previous language.
Write once, compile anywhere (WOCA).	Write once, run anywhere / everywhere (WORA / WORE).
Allows procedural programming, functional programming, object-oriented programming	Strongly encourages an object-oriented programming paradigm .
Allows direct calls to native system libraries.	Call through the Java Native Interface and recently Java Native Access.
Exposes low-level system facilities.	Runs in a virtual machine.
Only provides object types and type names.	Is reflective, allowing metaprogramming and dynamic code generation at runtime.
Pointers, references, and pass-by-value are supported.	Primitive and reference data types always passed by value.
Supports classes, structs, and unions, and can allocate them on heap or stack.	Only supports classes, and allocates them on the heap. Java SE6 optimizes with escape analysis to allocate some objects on the stack.
The C++ Standard Library has a much more limited scope and functionality	The standard library has grown with each release.
Operator overloading for most operators.	The meaning of operators is generally immutable, but the + and += operators have been overloaded for Strings.
Full Multiple inheritance, including virtual inheritance.	From classes, only single inheritance is allowed. From Interfaces, Multiple inheritance is allowed.
No standard inline documentation mechanism. Third-party software (e.g. Doxygen) exists.	Javadoc standard documentation.
Supports the goto statement.	Supports labels with loops and statement blocks.

Question: Explain the structure of java program. Exam-2015

Answer:

Structure of a Java program

A Java program is a collection of classes. Each class is normally written in a separate file and the name of the file is the name of the class contained in the file, with the extension .java.

Java requires that at least one of the classes has a public static method called main. More precisely we need a method with the following *signature*:

public static void main(String[] args)

The word void has the same connotation as in C -- this function does not return a value. The argument to main is an array of strings (we'll look at strings and arrays in Java in more detail later)--this corresponds to the argument to the main function in a C program.

We don't need argc because in Java, unlike in C, it is possible to extract the length of an array passed to a function. Also, unlike C, the argument to main is compulsory. Of course, the name of the argument is not important: in place of args we could have used xyz or any other valid identifier.

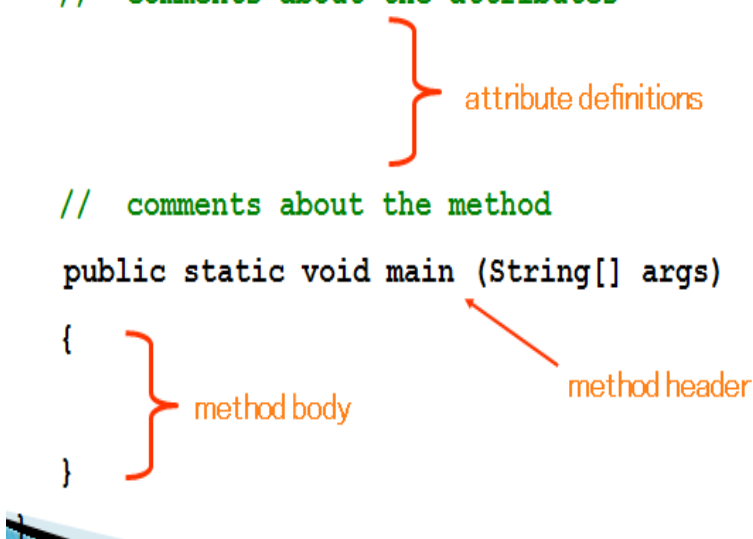
Here then, is a Java equivalent of the canonical *Hello world* program.

```
class helloworld{  
    public static void main(String[] args){  
        System.out.println("Hello world!");  
    }  
}
```

Here println is a static method in the class System.out which takes a string as argument and prints it out with a trailing newline character.

As noted earlier, we have to save this class in a file with the same name and the extension .java; that is, helloworld.java.

```
// comments about the class  
public class MyProgram  
{  
    // comments about the attributes  
    // comments about the method  
    public static void main (String[] args)  
    {  
    }  
}
```



Question: What is variable? What are the rules of variable naming?

Variables:

A variable is a name for a location in memory.

Or

A variable must be declared by specifying the variable's name and the type of information that it will hold

data type variable name

int total;

- ▶ Multiple variables can be created in one declaration:
- ▶ int count, temp, result;

Question: Write down the name of different primitive data types in java? Exam-2015

Data types:

- ▶ Primitive Data Types
- ▶ Variables, Initialization, and Assignment
- ▶ Constants
- ▶ Characters
- ▶ Strings

There are eight primitive data types in Java

- 📖 Four of them represent integers:
 - byte, short, int, long
- 📖 Two of them represent floating point numbers:
 - float, double
- 📖 One of them represents characters:
 - Char, String
- 📖 And one of them represents boolean values:
 - boolean

<u>Type</u>	<u>Storage</u>	<u>Min Value</u>	<u>Max Value</u>
byte	8 bits	-128	127
short	16 bits	-32,768	32,767
int	32 bits	-2,147,483,648	2,147,483,647
long	64 bits	$< -9 \times 10^{18}$	$> 9 \times 10^{18}$
float	32 bits	with 7 significant digits	
doubl	64 bits		
		with 15 significant digits	

Operator

An operator is a symbol that operates on one or more arguments to produce a result.

Operands

Operands are the values on which the operators act upon.

An operand can be:

- 📖 A numeric variable - integer, floating point or character
- 📖 Any primitive type variable - numeric and Boolean
- 📖 Reference variable to an object
- 📖 A literal - numeric value, Boolean value, or string.
- 📖 An array element, "a[2]"
- 📖 char primitive, which in numeric operations is treated as an unsigned two byte integer

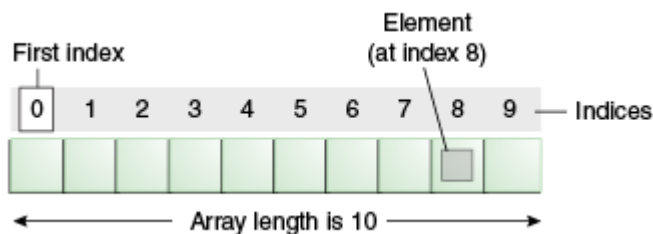
Types of Operators

- 📖 Assignment Operators
- 📖 Increment Decrement Operators
- 📖 Arithmetic Operators
- 📖 Bitwise Operators
- 📖 Relational Operators
- 📖 Logical Operators
- 📖 Ternary Operators

Question: What is an array? Write a Java program that can print the value 0 to 10 by using single dimension array. Exam-2014

Arrays:

Java provides a data structure, the array, which stores a **fixed-size sequential collection of elements of the same type**. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.



Declaring Array Variables

To use an array in a program, you must declare a variable to reference the array, and you must specify the type of array the variable can reference. Here is the syntax for declaring an array variable –

Syntax

```
dataType[] arrayRefVar;
```

Example

The following code snippets are examples of this syntax –

```
double[] myList; // preferred way.  
or  
double myList[]; // works but not preferred way.
```



Creating Arrays

You can create an array by using the new operator with the following syntax –

Syntax

```
arrayRefVar = new dataType[arraySize];
```

The above statement does two things –

-  It creates an array using new dataType[arraySize].
-  It assigns the reference of the newly created array to the variable arrayRefVar.







Declaring an array variable, creating an array, and assigning the reference of the array to the variable can be combined in one statement, as shown below –

```
dataType[] arrayRefVar = new dataType[arraySize];
```

Example

Following statement declares an array variable, myList, creates an array of 10 elements of double type and assigns its reference to myList –

```
double[] myList = new double[10];
```

```
 int a[]=new int[5]; //declaration and instantiation
 a[0]=10; //initialization
 a[1]=20;
 a[2]=70;
 a[3]=40;
 a[4]=50;
```

Following picture represents array myList. Here, myList holds ten double values and the indices are from 0 to 9.

Processing Arrays

When processing array elements, we often use either for loop or foreach loop because all of the elements in an array are of the same type and the size of the array is known.

Example

Here is a complete example showing how to create, initialize, and process arrays –

```
public class TestArray {

    public static void main(String[] args) {
        double[] myList = {1.9, 2.9, 3.4, 3.5};

        // Print all the array elements
        for (int i = 0; i < myList.length; i++) {
            System.out.println(myList[i] + " ");
        }

        // Summing all elements
        double total = 0;
        for (int i = 0; i < myList.length; i++) {
            total += myList[i];
        }
        System.out.println("Total is " + total);
    }
}
```



```
// Finding the largest element
double max = myList[0];
for (int i = 1; i < myList.length; i++) {
    if (myList[i] > max) max = myList[i];
}
System.out.println("Max is " + max);
}
```

The foreach Loops

Foreach loop enables us to traverse the complete array sequentially without using an index variable.

Example

The following code displays all the elements in the array myList –

```
public class TestArray {

    public static void main(String[] args) {
        double[] myList = {1.9, 2.9, 3.4, 3.5};

        // Print all the array elements
        for (double element: myList) {
            System.out.println(element);
        }
    }
}
```

Control flow:

The statements inside our source files are generally executed from top to bottom, in the order that they appear.

Control flow statements, however, break up the flow of execution by employing decision making, looping, and branching, enabling your program to conditionally execute particular blocks of code. This section describes the decision-making statements (if-then, if-then-else, switch), the looping statements (for, while, do-while), and the branching statements (break, continue, return) supported by the Java programming language.

Statement Type	Keyword
looping	while, do-while , for
decision making	if-else, switch-case
exception handling	try-catch-finally, throw
branching	break, continue, label:, return

The If Statement

The if statement executes a block of code only if the specified expression is true. If the value is false, then the if block is skipped and execution continues with the rest of the program.

You can either have a single statement or a block of code within an if statement. Note that the conditional expression must be a Boolean expression.

The simple if statement has the following syntax:

```
if (<conditional expression>)  
    <statement action>
```

Below is an example that demonstrates conditional execution based on if statement condition.

```
public class IfStatementDemo {  
  
    public static void main(String[] args) {  
        int a = 10, b = 20;  
        if (a > b)  
            System.out.println("a > b");  
        if (a < b)  
            System.out.println("b < a");  
    }  
}  
Output  
b > a
```

The for Statement

The for statement provides a compact way to iterate over a range of values.

The general form of the for statement can be expressed like this:

```
for (initialization; termination; increment) {  
    statement  
}
```

The switch Statement

Use the switch statement to conditionally perform statements based on an integer expression.

Following is a sample program, [SwitchDemo](#), that declares an integer named month whose value supposedly represents the month in a date. The program displays the name of the month, based on the value of month, using the switch statement:

```
public class SwitchDemo {  
    public static void main(String[] args) {  
        int month = 8;  
        switch (month) {  
            case 1: System.out.println("January"); break;  
            case 2: System.out.println("February"); break;  
            case 3: System.out.println("March"); break;  
            case 4: System.out.println("April"); break;  
            case 5: System.out.println("May"); break;  
            case 6: System.out.println("June"); break;  
            case 7: System.out.println("July"); break;  
            case 8: System.out.println("August"); break;  
            case 9: System.out.println("September"); break;  
            case 10: System.out.println("October"); break;  
        }  
    }  
}
```

```
        case 11: System.out.println("November"); break;
        case 12: System.out.println("December"); break;
    }
}
```

The while and do-while Statements

We use a *while* statement to continually execute a block of statements while a condition remains true. The general syntax of the while statement is:

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

First, the while statement evaluates *expression*, which must return a boolean value. If the expression returns true, then the while statement executes the statement(s) associated with it. The while statement continues testing the expression and executing its block until the expression returns false.

The example program shown below, called [WhileDemo](#) uses a while statement to step through the characters of a string, appending each character from the string to the end of a string buffer until it encounters the letter g.

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

        String copyFromMe = "Copy this string until you " +
            "encounter the letter 'g'.";
        StringBuffer copyToMe = new StringBuffer();

        int i = 0;
        char c = copyFromMe.charAt(i);

        while (c != 'g') {
            copyToMe.append(c);
            c = copyFromMe.charAt(++i);
        }
        System.out.println(copyToMe);
    }
}
```

Question: What is constructor?

Concept of Constructors:



A constructor is a special method that is used to initialize an object. A constructor is a class type function that automatically called when an object is created.

Constructor method has the same name as that of class, they are called or invoked when an object of class is created and can't be called explicitly. Attributes of an object may be available when creating objects if no attribute is available then default constructor is called, also some of the attributes may be known initially. It is optional to write constructor method in a class but due to their utility they are used.

A constructor has same name as the class in which it resides. Constructor in Java can not be abstract, static, final or synchronized. These modifiers are not allowed for constructor.

```
Class Car
{
String name ;
String model;
Car() //Constructor
{
name = "";
model = "";
}
}
```

There are two types of Constructor

-  Default Constructor
-  Parameterized constructor

Each time a new object is created at least one constructor will be invoked.

```
Car c = new Car() //Default constructor invoked
Car c = new Car(name); //Parameterized constructor invoked
```

Constructor Overloading

Like methods, a constructor can also be overloaded. Overloaded constructors are differentiated on the basis of their type of parameters or number of parameters. Constructor overloading is not much different than method overloading. In case of method overloading you have multiple methods with same name but different signature, whereas in Constructor overloading you have multiple constructor with different signature but only difference is that Constructor doesn't have return type in Java.

Question: Why do we Overload constructors?

Answer:

Constructor overloading is done to construct object in different ways.

Example of constructor overloading

```
Class Language {
String name;

Language () {
System.out.println("Constructor method called.");
}

Language(String t) {
name = t;
}

public static void main(String[] args) {
Language cpp = new Language();
Language java = new Language("Java");

cpp.setName("C++");
}
```



```
java.getName();
cpp.getName();
}

void setName(String t) {
    name = t;
}

void getName() {
    System.out.println("Language name: " + name);
}
}
```

Java application and Applet:

Java uses two styles of programming

-  Application
-  Applet

Java Application:

Java application programs run in a standalone environment with the support of a virtual machine (JVM), where as Java applets run in a browser and are subject to stringent security restrictions in terms of file and network access, where as an Java application can have free reign over these resources.

Java Applet:

An applet is a Java™ program designed to be included in an HTML Web document.

We can write your Java applet and include it in an HTML page, much in the same way an image is included. When you use a Java-enabled browser to view an HTML page that contains an applet, the applet's code is transferred to your system and is run by the browser's Java virtual machine.

The HTML document contains tags, which specify the name of the Java applet and its Uniform Resource Locator (URL). The URL is the location at which the applet bytecodes reside on the Internet. When an HTML document containing a Java applet tag is displayed, a Java-enabled Web browser downloads the Java bytecodes from the Internet and uses the Java virtual machine to process the code from within the Web document. These Java applets are what enable Web pages to contain animated graphics or interactive content.

You can also write a Java application that does not require the use of a Web browser.

For more information, see Writing Applets, Sun Microsystems' tutorial for Java applets. It includes an overview of applets, directions for writing applets, and some common applet problems.

Applications are stand-alone programs that do not require the use of a browser. Java applications run by starting the Java interpreter from the command line and by specifying the file that contains the compiled application. Applications usually reside on the system on which they are deployed. Applications access resources on the system, and are restricted by the Java security model.

Simple Java application example:

```
// HelloWorld.java
// A very simple Java Application

public class HelloWorld {
```

```
        public static void main( String args[] )
        {
            System.out.println( "Hello World! \n");
        }
    }
```

To execute the code you need to compile it, which produces a file HelloWorld.class
javac HelloWorld.java

You can now execute the program, remember to remove the .class extension when running
java HelloWorld

Simple Java Applet example:

Web Page code HelloWorld.html

```
<html>
<head></head>
<body>
    <applet code="HelloWorld.class" width=800 height=600></applet>
</body>
</html>
```

The Java Applet Code HelloWorld.java, remember you need to compile before
accessing the ## web page

```
// HelloWorld.java
// A very simple Java Applet
```

```
import java.awt.*;           // import class Graphics
import java.applet.*;        // import class JApplet
```

```
public class HelloWorld extends Applet {
    public void paint( Graphics g )
    {
        g.drawString( "Hello World!", 25, 25 );
    }
}
```

To execute the Java applet, point your browser to the html file

Important Question



1. Define method? Write down the name of all the parts of method header and explain each part of it. Exam-2014
2. What are the difference between constructors and methods? **Exam-2015**
3. What kind of inheritance support in Java? Give an example that shows implementation of inheritance in Java. Exam-2014
4. What is "Called method" and "Calling method"? Give an example to demonstrate calling a method. Exam-2014
5. Why is the main method declared as static? Exam-2013
6. What is method overriding? Explain with an example. Exam-2013
7. What is Java method? Give a general syntax of it. State how to define a method and describe each part of method header. Exam-2013
8. What is method overloading? Why is used in Java? **Exam-2015**

Question: Define method? Write down the name of all the parts of method header and explain each part of it. Exam-2014 or

Question: What is Java method? Give a general syntax of it. State how to define a method and describe each part of method header. Exam-2013

Methods:

A Java method is a collection of statements that are grouped together to perform an operation.

When you call the `System.out.println()` method, for example, the system actually executes several statements in order to display a message on the console.

Creating Method

Considering the following example to explain the syntax of a method –
Method definition consists of a method header and a method body. The same is shown in the following syntax –

General syntax of Method:

Syntax

```
modifier returnType nameOfMethod (Parameter List) {  
    // method body  
}
```

The syntax shown above includes –

modifier – It defines the access type of the method and it is optional to use.

returnType – Method may return a value.

nameOfMethod – This is the method name. The method signature consists of the method name and the parameter list.

Parameter List – The list of parameters, it is the type, order, and number of parameters of a method. These are optional, method may contain zero parameters.

Method body – the method body defines what the method does with the statements.

Example:

```
public static int methodName(int a, int b) {  
    // body  
}
```

Name of all part of method:

public static – modifier

int – return type

methodName – name of the method

a, b – formal parameters

int a, int b – list of parameters

Question: What is “Called method” and “Calling method”? Give an example to demonstrate calling a method. Exam-2014

Calling Method: The calling method is the method that contains the actual call.

Called Method: The called method is the method being called.

They are also called the Caller and the Calling methods.

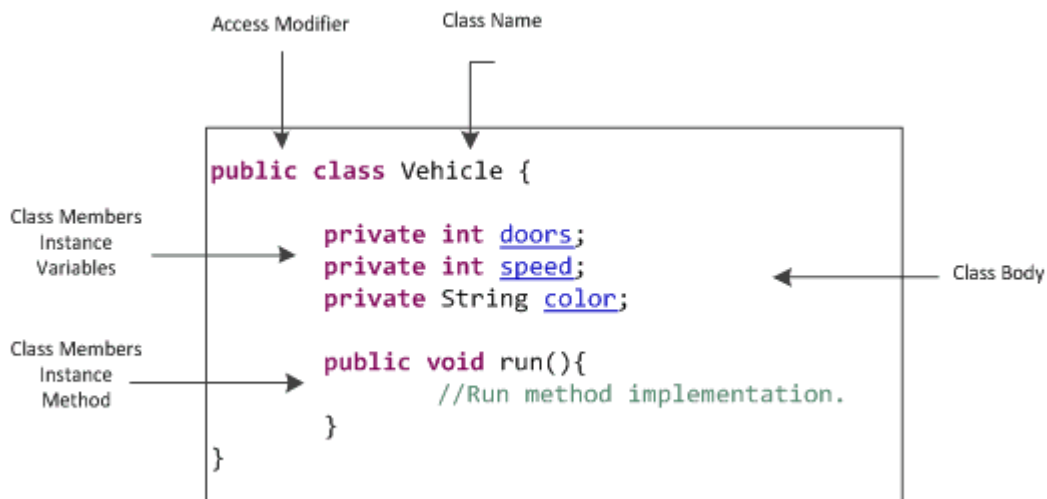
For exampl

```
// Calling method
void f()
{
    g();
}

// Called method
void g()
{
    Return 5;
}
```

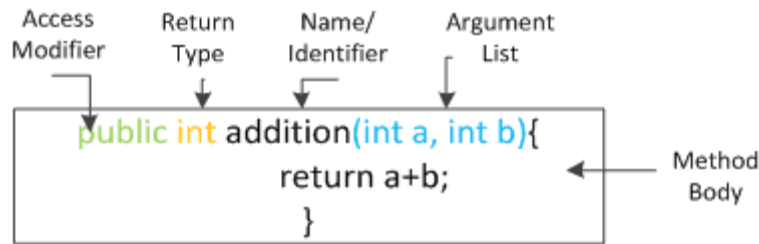
Declaring a class method:

Declaring a Class:



Declaring a Method:

A method is a program module that contains a series of statements that carry out a task. To execute a method, you invoke or call it from another method; the calling method makes a method call, which invokes the called method.



Calling a class method:

If you define the method as static you can use it without instantiating the class first, but then you also don't have the object variables available for use. In that case you could call it with `Foo.Bar()`. If it's a static method, you can call it by using the class name in place of an object.

Passing Parameters:

One of the first things we learn about a programming language we just laid hands on is the way it manages the parameters when we call a method. The two most common ways that languages deal with this problem are called “*passing by value*” and “*passing by reference*”.

Passing by value means that, whenever a call to a method is made, the parameters are evaluated, and the result value is copied into a portion of memory. When the parameter is used inside the method, either for read or write, we are actually using the copy, not the original value which is unaffected by the operations inside the method.

On the other hand, when a programming language uses *passing by reference*, the changes over a parameter inside a method will affect the original value. This is because what the method is receiving is the reference, i.e. the memory address, of the variable.

Some programming languages support *passing by value*, some support *passing by reference*, and some others support both. So the question is, what does Java support?

Local variables:

In computer science, a **local variable** is a **variable** that is given **local** scope. **Local variable** references in the function or block in which it is declared override the same **variable** name in the larger scope.

Variable scope:

A scope is a region of the program and broadly speaking there are three places, where variables can be declared – Inside a function or a block which is called local variables, In the definition of function parameters which is called formal parameters. Outside of all functions which is called global variables.

Question: Why is the main method declared as static? Exam-2013

Answer:

This is necessary because `main()` is called by the JVM before any objects are made. Since it is **static** it can be directly invoked via the class. Similarly, we use **static** sometime for user defined **methods** so that we need not to make objects. `void` indicates that the **main() method** being **declared** does not return a value.

Question: What is method overriding? Explain with an example. Exam-2013**Method overriding:**

In object-oriented programming, is a language feature that allows a subclass or child class to provide a specific implementation of a **method** that is already provided by one of its superclasses or parent classes.

Example:

In Java, when a subclass contains a method that overrides a method of the superclass, it can also invoke the superclass method by using the keyword `super` Example:

```
class Thought {
    public void message() {
        System.out.println("I feel like I am diagonally parked in a parallel
universe.");
    }
}

public class Advice extends Thought {
    @Override // @Override annotation in Java 5 is optional but helpful.
    public void message() {
        System.out.println("Warning: Dates in calendar are closer than they
appear.");
    }
}
```

Class `Thought` represents the superclass and implements a method call `message()`. The subclass called `Advice` inherits every method that could be in the `Thought` class. However, class `Advice` overrides the method `message()`, replacing its functionality from `Thought`.

```
Thought parking = new Thought();
parking.message(); // Prints "I feel like I am diagonally parked in a parallel universe."
```

```
Thought dates = new Advice(); // Polymorphism
dates.message(); // Prints "Warning: Dates in calendar are closer than they appear."
```

The `super` reference can be



```
public class Advice extends Thought {
    @Override
    public void message() {
        System.out.println("Warning: Dates in calendar are closer than they appear.");
        super.message(); // Invoke parent's version of method.
    }
}
```

There are methods that a subclass cannot override. For example, in Java, a method that is declared `final` in the super class cannot be overridden. Methods that are declared `private` or `static` cannot be overridden either because they are implicitly `final`. It is also impossible for a class that is declared `final` to become a super class.

Question: What is of inheritance support in Java? Give an example that shows implementation of inheritance in Java. Exam-2014**Implementation of inheritance:**

Inheritance in java is a mechanism in which one object acquires all the properties and behaviors of parent object.

Why use inheritance in java

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






Syntax of Java Inheritance

```
class Subclass-name extends Superclass-name
{
    //methods and fields
}
```

The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

Types of inheritance in java:

On the basis of class, there can be three types of inheritance in java:

-  Single Inheritance
-  Multiple Inheritance (**Through Interface**)
-  Multilevel Inheritance
-  Hierarchical Inheritance
-  Hybrid Inheritance (**Through Interface**)

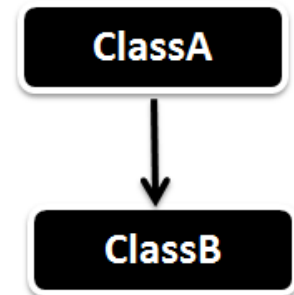
Single Inheritance in Java

Single Inheritance is the simple inheritance of all, When a class extends another class(Only one class) then we call it as Single inheritance.

The below diagram represents the single inheritance in java where **Class B** extends only one class **Class A**. Here **Class B** will be the **Sub class** and **Class A** will be one and only **Super class**.

```
class A
{
    public void methodA()
    {
        System.out.println("Base class method");
    }
}

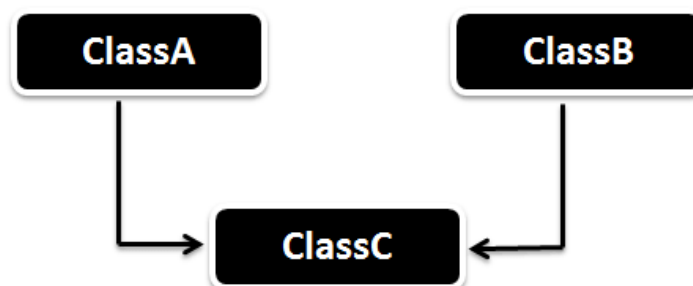
Class B extends A
{
    public void methodB()
    {
        System.out.println("Child class method");
    }
    public static void main(String args[])
    {
        B obj = new B();
        obj.methodA(); //calling super class method
        obj.methodB(); //calling local method
    }
}
```



Multiple Inheritance

“Multiple Inheritance” refers to the concept of one class extending (Or inherits) more than one base class.

The inheritance we learnt earlier had the concept of one base class or parent. The problem with “multiple inheritance” is that the derived class will have to manage the dependency on two base classes.

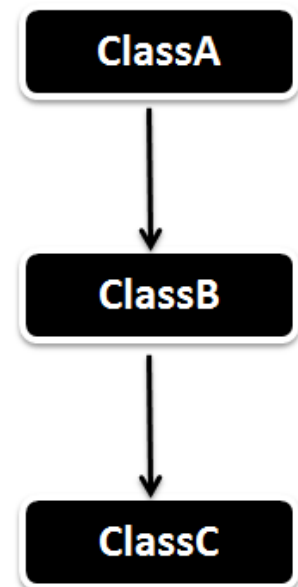


Multilevel Inheritance

Multilevel inheritance refers to a mechanism in OO technology where one can inherit from a derived class, thereby making this derived class the base class for the new class.

As you can see in below flow diagram C is subclass or child class of B and B is a child class of A. For more details and example refer – Multilevel inheritance in Java.

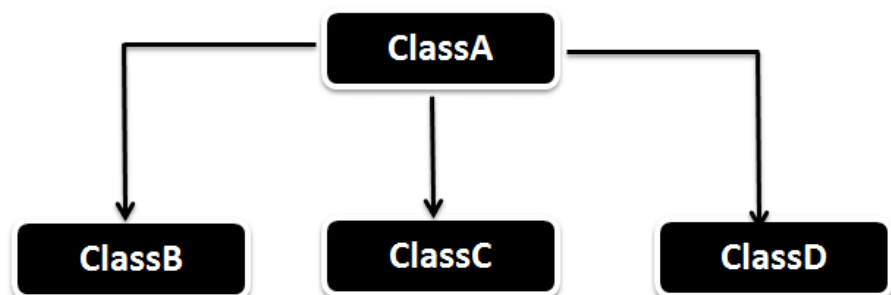
```
Class X
{
    public void methodX()
    {
        System.out.println("Class X method");
    }
}
Class Y extends X
{
    public void methodY()
    {
        System.out.println("class Y method");
    }
}
Class Z extends Y
{
    public void methodZ()
    {
        System.out.println("class Z method");
    }
    public static void main(String args[])
    {
        Z obj = new Z();
        obj.methodX(); //calling grand parent class method
        obj.methodY(); //calling parent class method
        obj.methodZ(); //calling local method
    }
}
```



Hierarchical Inheritance

In Hierarchical inheritance one parent class will be inherited by many sub classes. As per the below example ClassA will be inherited by ClassB, ClassC and ClassD. ClassA will be acting as a parent class for ClassB, ClassC and ClassD.

In below example class B,C and D inherits the same class A. A is parent class (or base class) of B,C & D.

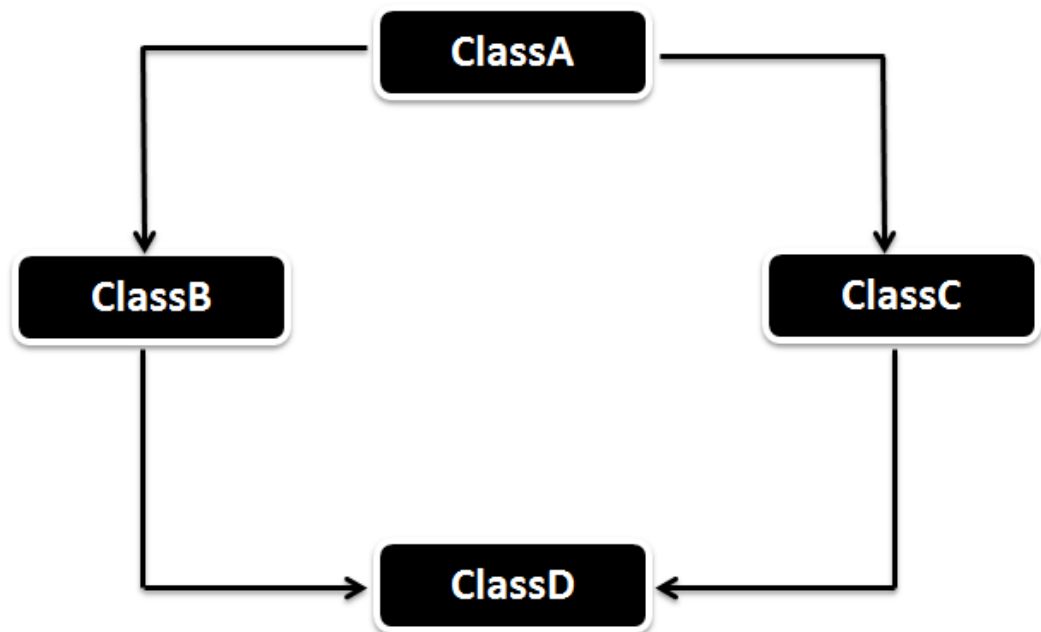



```
class A
{
    public void methodA()
    {
        System.out.println("method of Class A");
    }
}
class B extends A
{
    public void methodB()
    {
        System.out.println("method of Class B");
    }
}
class C extends A
{
    public void methodC()
    {
        System.out.println("method of Class C");
    }
}
class D extends A
{
    public void methodD()
    {
        System.out.println("method of Class D");
    }
}
class JavaExample
{
    public static void main(String args[])
    {
        B obj1 = new B();
        C obj2 = new C();
        D obj3 = new D();
        //All classes can access the method of class A
        obj1.methodA();
        obj2.methodA();
        obj3.methodA();
    }
}
Output:
method of Class A
method of Class A
method of Class A
```

Hybrid Inheritance

In simple terms you can say that Hybrid inheritance is a combination of Single and Multiple inheritance. A typical flow diagram would look like below. A hybrid inheritance can be achieved in the java in a same way as multiple inheritance can be!!

Using interfaces. yes you heard it right. By using **interfaces** you can have multiple as well as **hybrid inheritance** in Java.



```
class C
{
    public void disp()
    {
        System.out.println("C");
    }
}

class A extends C
{
    public void disp()
    {
        System.out.println("A");
    }
}

class B extends C
{
    public void disp()
    {
        System.out.println("B");
    }
}

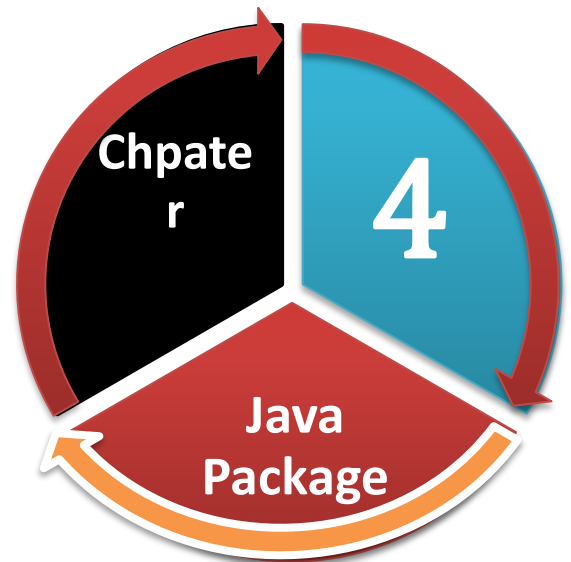
class D extends A
{
    public void disp()
    {
```

```
        System.out.println("D");
    }
    public static void main(String args[]){

        D obj = new D();
        obj.disp();
    }
}
```

Output:
D

Important Question



1. What is the difference between container and component in a GUI? **Exam-2015**
2. Write down the constructors of Flow Layout, Card Layout and GridBag Layout manager. Explain them. **Exam-2015**
3. Describe how to handle MouseEvent and KeyEvent in Java? **Exam-2015**
4. Define top-level and secondary container. Write about flow layout with syntax used in Java. **Exam-2014**
5. What is an event? List down the names of some of the Java AWT event listener interfaces. How to implement a listener interface. **Exam-2013**
6. Define actionPerformed(). What is the signature of the actionPerformed method? **Exam-2014**
7. What is layout manager? Briefly explain about the basic layout manager in Java and give the syntax of each layout manager. **Exam-2013**

Using Standard Java Packages:

Question: What is a user interface?

Answer:

The user interface is that part of a program that interacts with the user of the program. User interfaces take many forms. These forms range in complexity from simple command-line interfaces to the point-and-click graphical user interfaces provided by many modern applications

There are two types of GUI elements:

Component: Components are elementary GUI entities, such as Button, Label, and TextField.

Container: Containers, such as Frame and Panel, are used to hold components in a specific layout (such as FlowLayout or GridLayout). A container can also hold sub-containers.

AWT Packages

AWT is huge! It consists of 12 packages of 370 classes (Swing is even bigger, with 18 packages of 737 classes as of JDK 1.8). Fortunately, only 2 packages - java.awt and java.awt.event - are commonly-used.

The java.awt package contains the core AWT graphics classes:

- 📖 GUI Component classes, such as Button, TextField, and Label,
- 📖 GUI Container classes, such as Frame and Panel,
- 📖 Layout managers, such as FlowLayout, BorderLayout and GridLayout,
- 📖 Custom graphics classes, such as Graphics, Color and Font.

The java.awt.event package supports event handling:

- 📖 Event classes, such as ActionEvent, MouseEvent, KeyEvent and WindowEvent,
- 📖 Event Listener Interfaces, such as ActionListener, MouseListener, KeyListener and WindowListener,
- 📖 Event Listener Adapter classes, such as MouseAdapter, KeyAdapter, and WindowAdapter.
- 📖 AWT provides a platform-independent and device-independent interface to develop graphic programs that runs on all platforms, including Windows, Mac OS, and Unixes.

2.3 AWT Container Classes:

Question: Define top-level and secondary container .Write about flow layout with syntax used in Java. Exam-2014

Top-Level Containers: Frame, Dialog and Applet

Each GUI program has a *top-level container*. The commonly-used top-level containers in AWT are Frame, Dialog and Applet:

- 📖 A Frame provides the "main window" for the GUI application, which has a title bar (containing an icon, a title, the minimize, maximize/restore-down and close buttons), an optional menu bar, and the content display area. To write a GUI program, we typically start with a subclass extending from java.awt.Frame to inherit the main window as follows:

Secondary Containers: Panel and ScrollPane

Secondary containers are placed inside a top-level container or another secondary container. AWT also provide these secondary containers:

- 📖 Panel: a rectangular box under a higher-level container, used to *layout* a set of related GUI components in pattern such as grid or flow.
- 📖 ScrollPane: provides automatic horizontal and/or vertical scrolling for a single child component.

Basic Terminologies

Term	Description
Component	Component is an object having a graphical representation that can be displayed on the screen and that can interact with the user. For examples buttons, checkboxes, list and scrollbars of a graphical user interface.
Container	The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container Examples: panel, box
Frame	A Frame is a top-level window with a title and a border. The size of the frame includes any area designated for the border. Frame encapsulates window. It and has a title bar, menu bar, borders, and resizing corners.
Panel	Panel provides space in which an application can attach any other components, including other panels.
Window	Window is a rectangular area which is displayed on the screen. In different window we can execute different program and display different data. Window provide us with multitasking environment. A window must have either a frame, dialog, or another window defined as its owner when it's constructed.

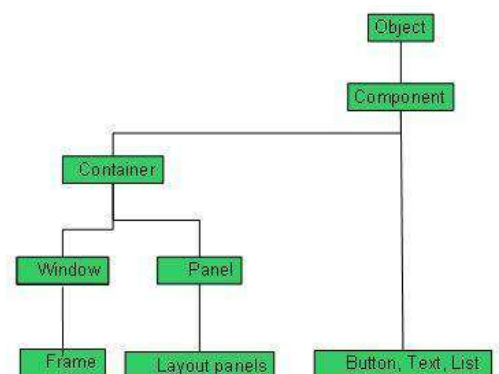
Useful Methods of Component class

Every user interface considers the following three main aspects:

UI elements: These are the core visual elements the user eventually sees and interacts with. GWT provides a huge list of widely used and common elements varying from basic to complex which we will cover in this tutorial.

Layouts: They define how UI elements should be organized on the screen and provide a final look and feel to the GUI (Graphical User Interface). This part will be covered in Layout chapter.

Behavior: These are events which occur when the user interacts with UI elements. This part will be covered in Event Handling chapter.



AWT UI Elements:

Following is the list of commonly used controls while designed GUI using AWT.

Sr. No.	Control & Description
1	Label: A Label object is a component for placing text in a container.
2	Button: This class creates a labeled button.
3	Check Box: A check box is a graphical component that can be in either an on (true) or off (false) state.
4	Check Box Group: The CheckboxGroup class is used to group the set of checkbox.
5	List: The List component presents the user with a scrolling list of text items.
6	Text Field: A TextField object is a text component that allows for the editing of a single line of text.
7	Text Area: A TextArea object is a text component that allows for the editing of a multiple lines of text.
8	Choice: A Choice control is used to show pop up menu of choices. Selected choice is shown on the top of the menu.
9	Canvas: A Canvas control represents a rectangular area where application can draw something or can receive inputs created by user.
10	Image: An Image control is superclass for all image classes representing graphical images.
11	Scroll Bar: A Scrollbar control represents a scroll bar component in order to enable user to select from range of values.
12	Dialog: A Dialog control represents a top-level window with a title and a border used to take some form of input from the user.
13	File Dialog: A FileDialog control represents a dialog window from which the user can select a file.

Question: What is the difference between container and component in a GUI? Exam-2015

Answer:

Difference between container and component:

Container	Component
A container is a component that holds and manages other components. Containers display components using a layout manager.	a component is the basic user interface object and is found in all Java applications. Components include lists, buttons, panels, and windows.
To use components, we need to place them in a container.	Component important for interaction
JComponent, in turn, inherits from the Container class in the Abstract Windowing Toolkit (AWT). So Swing is based on classes inherited from AWT.	Swing components inherit from the javax.Swing.JComponent class, which is the root of the Swing component hierarchy.
Containers: Frame Window Dialog Panel	List of common Components List Scrollbar TextArea TextField Choice Button

	Label

Creating graphical user interfaces with AWT.

To create simple awt example, you need a frame. There are two ways to create a frame in AWT.

1. By extending Frame class (inheritance)
2. By creating the object of Frame class (association)

Frame Typically holds (hosts) other components Common methods:

- 📖 JFrame(String title) – constructor, title optional
- 📖 setSize(int width, int height) – set size
- 📖 add(Component c) – add component to window
- 📖 setVisible(boolean v) – make window visible or not. Don't forget this public void
- 📖 setDefaultCloseOperation(int op) Makes the frame perform the given action when it closes.
 - Common value passed: JFrame.EXIT_ON_CLOSE
 - If not set, the program will never exit even if the frame is closed.
- 📖 public void setSize(int width, int height) Gives the frame a fixed size in pixels.
- 📖 public void pack() Resizes the frame to fit the components inside it snugly.

Some Attribute of Frame:

name	description
background	background color behind component
border	border line around component
enabled	whether it can be interacted with
focusable	whether key text can be typed on it
font	font used for text in component
foreground	foreground color of component
height, width	component's current size in pixels
visible	whether component can be seen
tooltip text	text shown when hovering mouse
size, minimum / maximum / preferred size	various sizes, size limits, or desired sizes that the component may take

AWT Example by Inheritance

Let's see a simple example of AWT where we are inheriting Frame class. Here, we are showing Button component on the Frame.

```

1. import java.awt.*;
2. class First extends Frame{
3.     First(){
4.         Button b=new Button("click me");
5.         b.setBounds(30,100,80,30); // setting button position
6.         add(b); // adding button into frame
7.         setSize(300,300); // frame size 300 width and 300 height
8.         setLayout(null); // no layout manager
9.         setVisible(true); // now frame will be visible, by default not visible
10.    }
11.    public static void main(String args[]){
12.        First f=new First();
13.    }

```

AWT Example by Association

Let's see a simple example of AWT where we are creating instance of Frame class. Here, we are showing Button component on the Frame.

```

import java.awt.*;
class First2{
    First2(){
        Frame f=new Frame();
        Button b=new Button("click me");
        b.setBounds(30,50,80,30);
        f.add(b);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[]){
        First2 f=new First2();
    }
}

```

Method	Description
public void add(Component c)	inserts a component on this component.
public void setSize(int width,int height)	sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	defines the layout manager for the component.
public void setVisible(boolean status)	changes the visibility of the component, by default

false.

Managing graphics objects with GUI layout managers.

Question: What is layout manager? Briefly explain about the basic layout manager in Java and give the syntax of each layout manager. Exam-2013

Question: Write down the constructors of Flow Layout, Card Layout and GridBag Layout manager. Explain them. Exam-2015

Layout Manager:

A layout manager is an object that implements the `LayoutManager` interface* and determines the size and position of the components within a container.

Although components can provide size and alignment hints, a container's layout manager has the final say on the size and position of the components within the container.

The `LayoutManagers` are used to arrange components in a particular manner. `LayoutManager` is an interface that is implemented by all the classes of layout managers.

There are following classes that represents the layout managers:

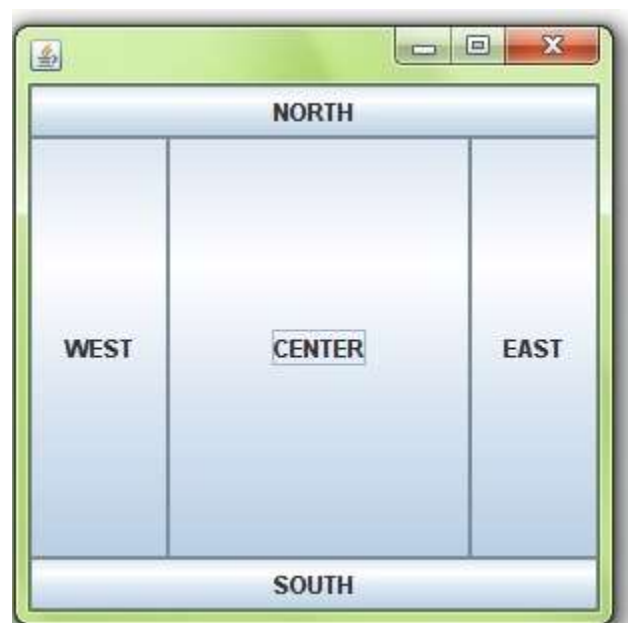
- 📖 `java.awt.BorderLayout`
- 📖 `java.awt.FlowLayout`
- 📖 `java.awt.GridLayout`
- 📖 `java.awt.CardLayout`
- 📖 `java.awt.GridBagLayout`
- 📖 `javax.swing.BoxLayout`
- 📖 `javax.swing.GroupLayout`
- 📖 `javax.swing.ScrollPaneLayout`
- 📖 `javax.swing.SpringLayout` etc.

We Discuss here some AWT of layout:

📖 **Java BorderLayout**

The `BorderLayout` is used to arrange the components in five regions: north, south, east, west and center. Each region (area) may contain one component only. It is the default layout of frame or window. The `BorderLayout` provides five constants for each region:

1. `public static final int NORTH`
2. `public static final int SOUTH`
3. `public static final int EAST`
4. `public static final int WEST`
5. `public static final int CENTER`



```

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

public class Border {
    JFrame f;
    Border(){
        f=new JFrame();

        JButton b1=new JButton("NORTH");
        JButton b2=new JButton("SOUTH");
        JButton b3=new JButton("EAST");
        JButton b4=new JButton("WEST");
        JButton b5=new JButton("CENTER");

        f.add(b1, BorderLayout.NORTH);
        f.add(b2, BorderLayout.SOUTH);
        f.add(b3, BorderLayout.EAST);
        f.add(b4, BorderLayout.WEST);
        f.add(b5, BorderLayout.CENTER);

        f.setSize(300,300);
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new Border();
    } }

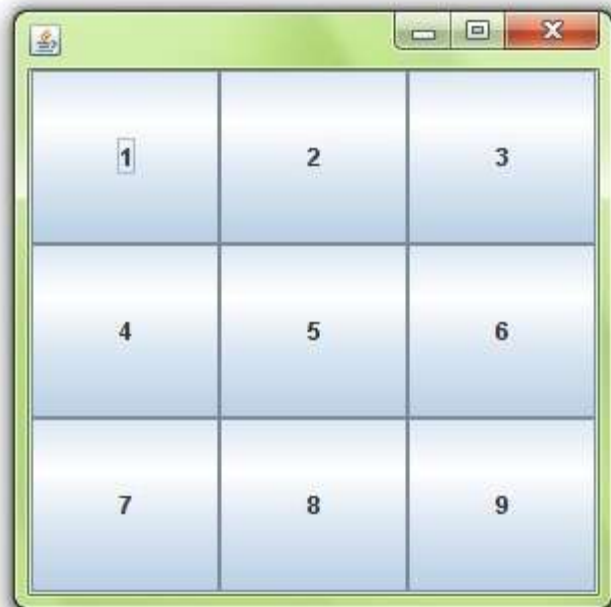
```

Java GridLayout

The GridLayout is used to arrange the components in rectangular grid. One component is displayed in each rectangle.

Constructors of GridLayout class

1. **GridLayout()**: creates a grid layout with one column per component in a row.
2. **GridLayout(int rows, int columns)**: creates a grid layout with the given rows and columns but no gaps between the components.
3. **GridLayout(int rows, int columns, int hgap, int vgap)**: creates a grid layout with the given rows and columns alongwith given horizontal and vertical gaps.



Example of GridLayout class

```

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

public class MyGridLayout{

```

```
JFrame f;  
MyGridLayout(){  
    f=new JFrame();  
  
    JButton b1=new JButton("1");  
    JButton b2=new JButton("2");  
    JButton b3=new JButton("3");  
    JButton b4=new JButton("4");  
    JButton b5=new JButton("5");  
        JButton b6=new JButton("6");  
        JButton b7=new JButton("7");  
    JButton b8=new JButton("8");  
        JButton b9=new JButton("9");  
  
    f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);  
    f.add(b6);f.add(b7);f.add(b8);f.add(b9);  
  
    f.setLayout(new GridLayout(3,3));  
    //setting grid layout of 3 rows and 3 columns  
  
    f.setSize(300,300);  
    f.setVisible(true);  
}  
public static void main(String[] args) {  
    new MyGridLayout();  
}  
}
```

Java FlowLayout

The FlowLayout is used to arrange the components in a line, one after another (in a flow). It is the default layout of applet or panel.

Fields of FlowLayout class

1. **public static final int LEFT**
2. **public static final int RIGHT**
3. **public static final int CENTER**
4. **public static final int LEADING**
5. **public static final int TRAILING**

Constructors of FlowLayout class

1. **FlowLayout()**: creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
2. **FlowLayout(int align)**: creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
3. **FlowLayout(int align, int hgap, int vgap)**: creates a flow layout with the given alignment and the given horizontal and vertical gap.

Example of FlowLayout class

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

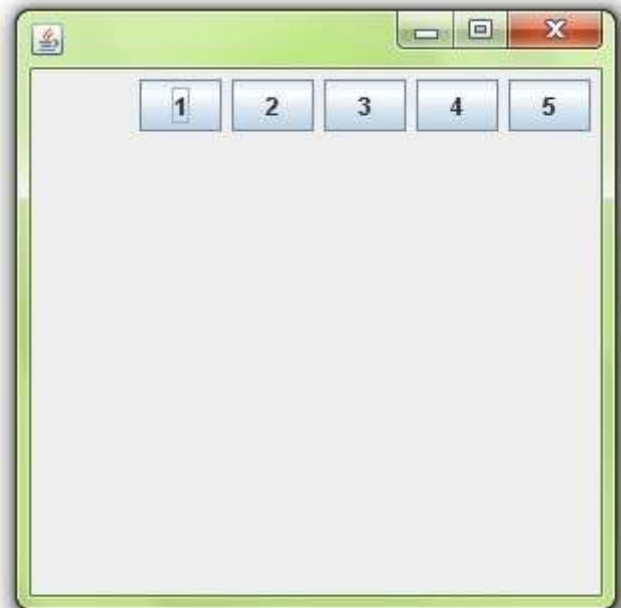
public class MyFlowLayout{
    JFrame f;
    MyFlowLayout(){
        f=new JFrame();

        JButton b1=new JButton("1");
        JButton b2=new JButton("2");
        JButton b3=new JButton("3");
        JButton b4=new JButton("4");
        JButton b5=new JButton("5");

        f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);

        f.setLayout(new FlowLayout(FlowLayout.RIGHT));
        //setting flow layout of right alignment

        f.setSize(300,300);
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new MyFlowLayout();
    }
}
```



Java BoxLayout

The BorderLayout is used to arrange the components either vertically or horizontally. For this purpose, BorderLayout provides four constants. They are as follows:

Note: BorderLayout class is found in javax.swing package.

Fields of BorderLayout class

1. **public static final int X_AXIS**
2. **public static final int Y_AXIS**
3. **public static final int LINE_AXIS**
4. **public static final int PAGE_AXIS**

Example of BorderLayout class with Y-

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

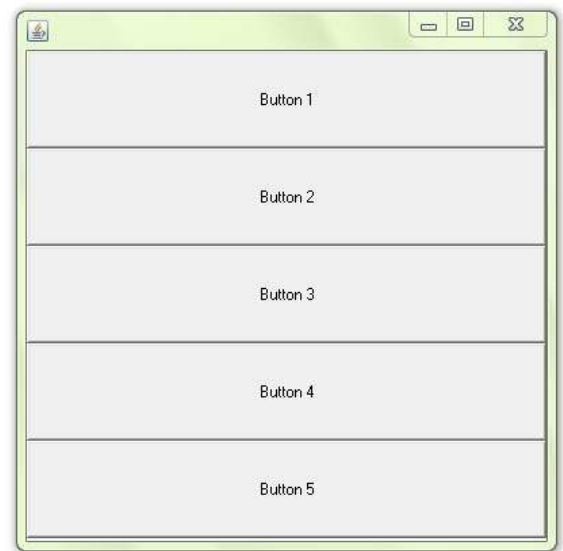
public class BorderLayoutExample1 extends Frame {
    Button buttons[];

    public BorderLayoutExample1 () {
        buttons = new Button [5];

        for (int i = 0;i<5;i++) {
            buttons[i] = new Button ("Button " + (i + 1));
            add (buttons[i]);
        }

        setLayout (new BorderLayout (this, BorderLayout.Y_AXIS)
        );
        setSize(400,400);
        setVisible(true);
    }

    public static void main(String args[]){
        BorderLayoutExample1 b=new BorderLayoutExample1();
    }
}
```



Java CardLayout

The CardLayout class manages the components in such a manner that only one component is visible at a time. It treats each component as a card that is why it is known as CardLayout. Constructors of CardLayout class

1. **CardLayout()**: creates a card layout with zero horizontal and vertical gap.
2. **CardLayout(int hgap, int vgap)**: creates a card layout with the given horizontal and vertical gap.

Commonly used methods of CardLayout class

- 📖 **public void next(Container parent)**: is used to flip to the next card of the given container.
- 📖 **public void previous(Container parent)**: is used to flip to the previous card of the given container.
- 📖 **public void first(Container parent)**: is used to flip to the first card of the given container.

- 📖 **public void last(Container parent):** is used to flip to the last card of the given container.
- 📖 **public void show(Container parent, String name):** is used to flip to the specified card with the given name.

Example of CardLayout class

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

import javax.swing.*;

public class CardLayoutExample extends JFrame
implements ActionListener{
    CardLayout card;
    JButton b1,b2,b3;
    Container c;
    CardLayoutExample(){

        c=getContentPane();
        card=new CardLayout(40,30);
        //create CardLayout object with 40 hor space
        and 30 ver space
        c.setLayout(card);

        b1=new JButton("Apple");
        b2=new JButton("Boy");
        b3=new JButton("Cat");
        b1.addActionListener(this);
        b2.addActionListener(this);
        b3.addActionListener(this);

        c.add("a",b1);c.add("b",b2);c.add("c",b3);

    }
    public void actionPerformed(ActionEvent e) {
        card.next(c);
    }

    public static void main(String[] args) {
        CardLayoutExample cl=new CardLayoutExample();
        cl.setSize(400,400);
        cl.setVisible(true);
        cl.setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}
```



📖 Java GridBagLayout

The Java GridBagLayout class is used to align components vertically, horizontally or along their baseline.

The components may not be of same size. Each GridBagLayout object maintains a dynamic, rectangular grid of cells. Each component occupies one or more cells known as its display area. Each component associates an instance of GridBagConstraints. With the help of constraints object we arrange component's display area on the grid. The

GridBagLayout manages each component's minimum and preferred sizes in order to determine component's size.

Summary of Layout Manager:

Sr. No.	Layout Manager & Description
1	BorderLayout The BorderLayout arranges the components to fit in the five regions: east, west, north, south and center.
3	FlowLayout The FlowLayout is the default layout. It layouts the components in a directional flow.
4	GridLayout The GridLayout manages the components in form of a rectangular grid.
5	GridBagLayout This is the most flexible layout manager class. The object of GridBagLayout aligns the component vertically, horizontally or along their baseline without requiring the components of same size.

Drawing methods include:

- 📖 drawString – For drawing text
- 📖 g.drawString("Hello", 10, 10);
- 📖 drawImage – For drawing images
- 📖 g.drawImage(img,
0, 0, width, height,
0, 0, imageWidth, imageHeight,
null);

drawLine, drawArc, drawRect, drawOval, drawPolygon – For drawing geometric shapes

g2.draw(new Line2D.Double(0, 0, 30, 40));

Depending on your current need, you can choose one of several methods in the Graphics class based on the following criteria:

- 📖 Whether you want to render the image at the specified location in its original size or scale it to fit inside the given rectangle
- 📖 Whether you prefer to fill the transparent areas of the image with color or keep them transparent

Fill methods apply to geometric shapes and include fillArc, fillRect, fillOval, fillPolygon.

Whether you draw a line of text or an image, remember that in 2D graphics every point is determined by its x and y coordinates. All of the draw and fill methods need this information which determines where the text or image should be rendered.

For example, to draw a line, an application calls the following:

```
java.awt.Graphics.drawLine(int x1, int y1, int x2, int y2)
```


In this code $(x1, y1)$ is the start point of the line, and $(x2, y2)$ is the end point of the line.

So the code to draw a horizontal line is as follows:

```
Graphics.drawLine(20, 100, 120, 100);
```

Each Graphics object has its own coordinate system, and all the methods of Graphics including those for drawing Strings, lines, rectangles, circles, polygons and more. Drawing in Java starts with particular Graphics object. You get access to the Graphics object through the `paint(Graphics g)` method of your applet. Each draw method call will look like `g.drawString("Hello World", 0, 50)` where `g` is the particular Graphics object with which you're drawing.

For convenience's sake in this lecture the variable `g` will always refer to a preexisting object of the Graphics class. As with any other method you are free to use some other name for the particular Graphics context, `myGraphics` or `appletGraphics` perhaps.

Layout strategy for border interactors

As an example, suppose that you want to write a program that displays two buttons—**Start** and **Stop**—at the bottom of a program window. Let's ignore for the moment what those buttons actually do and concentrate instead on how to make them appear. If you use the standard layout management tools provided by the **Program** class, all you have to do is include the following code as part of the **init** method:

```
add(new JButton("Start"), SOUTH);  
add(new JButton("Stop"), SOUTH);
```

Question: How to implement a listener interface. Exam-2013

Answer:

Assigning action listeners to the buttons

Creating the buttons, however, accomplishes only part of the task. To make the buttons active, you need to give each one an action listener so that pressing the button performs the appropriate action.

These days, the most common programming style among experienced Java programmers is to assign an individual action listener to each button in the form of an anonymous inner class. Suppose, for example, that you want the Start and Stop buttons to invoke methods called `startAction` and `stopAction`, respectively. You could do so by changing the initialization code as follows:

```

JButton startButton = new JButton("Start");
startButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        startAction();
    }
});
add(startButton, SOUTH);
JButton stopButton = new JButton("Start");
stopButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        stopAction();
    }
});
add(stopButton, SOUTH);

```

Question: Define `actionPerformed()`. What is the signature of the `actionPerformed` method? Exam-2014

Answer:

Define `actionPerformed()`.

The `actionPerformed()` method is invoked automatically whenever you click on the registered component.

`public abstract void actionPerformed(ActionEvent e);`

The `ActionEvent` argument `e` is the object that represents the event. This object contains information about the event, like which component is the event source. The `ActionEvent` class has two methods that you might find beneficial: `getActionCommand()` and `getSource()` (which is actually inherited by the `ActionEvent` class from the `EventObject` class). The first method returns a string that identifies the action. This string is usually set by the event source, using a method called `setActionCommand`. The second method returns the event source object.

Example:

```

import java.awt.*;
import java.awt.event.*;

```

```
public class ActionListenerExample {  
    public static void main(String[] args) {  
        Frame f=new Frame("ActionListener Example");  
        final TextField tf=new TextField();  
        tf.setBounds(50,50, 150,20);  
        Button b=new Button("Click Here");  
        b.setBounds(50,100,60,30);  
  
        b.addActionListener(new ActionListener(){  
            public void actionPerformed(ActionEvent e){  
                tf.setText("Welcome to Javatpoint.");  
            }  
        });  
        f.add(b);f.add(tf);  
        f.setSize(400,400);  
        f.setLayout(null);  
        f.setVisible(true);  
    }  
}
```



Signature of the actionPerformed method:

The signature of actionPerformed method is `ActionEvent e`

Event handling of various components.

Question: What is an event? List down the names of some of the Java AWT event listener interfaces.

Question: What is an Event?

Change button in the state of an object is known as event.

i.e. event describes the change in state of source. Events are generated as result of user interaction with the graphical user interface components. For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page are the activities that causes an event to happen.

Types of Event

The events can be broadly classified into two categories:

Foreground Events

Those events which require the direct interaction of user.

They are generated as consequences of a person interacting with the graphical components in Graphical User Interface. For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page etc.

Background Events –

Those events that require the interaction of end user are known as background events.

Operating system interrupts, hardware or software failure, timer expires, an operation completion are the example of background events.

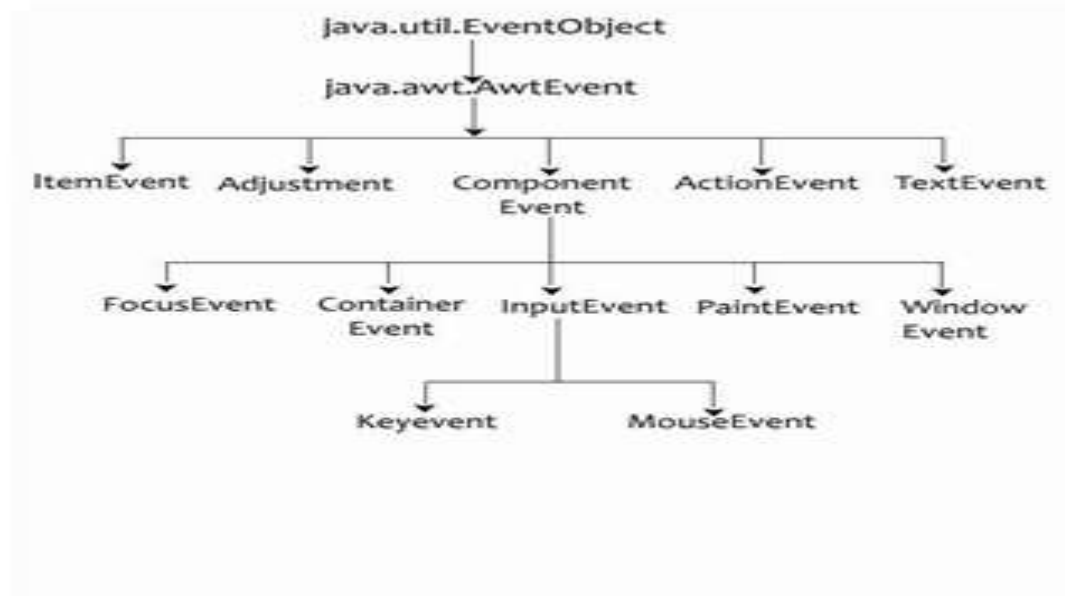
List Java AWT event listener interface:

Interface Summary	
Interface	Description
ActionListener	The listener interface for receiving action events.
AdjustmentListener	The listener interface for receiving adjustment events.
AWTEventListener	The listener interface for receiving notification of events dispatched to objects that are instances of Component or MenuComponent or their subclasses.
ComponentListener	The listener interface for receiving component events.
ContainerListener	The listener interface for receiving container events.
FocusListener	The listener interface for receiving keyboard focus events on a component.
HierarchyBoundsListener	The listener interface for receiving ancestor moved and resized events.
HierarchyListener	The listener interface for receiving hierarchy changed events.
InputMethodListener	The listener interface for receiving input method events.
ItemListener	The listener interface for receiving item events.
KeyListener	The listener interface for receiving keyboard events (keystrokes).
MouseListener	The listener interface for receiving "interesting" mouse events (press, release, click, enter, and exit) on a component.
MouseMotionListener	The listener interface for receiving mouse motion events on a component.
MouseWheelListener	The listener interface for receiving mouse wheel events on a component.
TextListener	The listener interface for receiving text events.
WindowFocusListener	The listener interface for receiving WindowEvents, including WINDOW_GAINED_FOCUS and WINDOW_LOST_FOCUS events.
WindowListener	The listener interface for receiving window events.
WindowStateListener	The listener interface for receiving window state events.

List Java AWT event listener classes:

Class Summary	
Class	Description
ActionEvent	A semantic event which indicates that a component-defined action occurred.
AdjustmentEvent	The adjustment event emitted by Adjustable objects like Scrollbar and ScrollPane .
AWTEventListenerProxy	A class which extends the EventListenerProxy specifically for adding an AWTEventListener for a specific event mask.
ComponentAdapter	An abstract adapter class for receiving component events.
ComponentEvent	A low-level event which indicates that a component moved, changed size, or changed visibility (also, the root class for the other component-level events).
ContainerAdapter	An abstract adapter class for receiving container events.
ContainerEvent	A low-level event which indicates that a container's contents changed because a component was added or removed.
FocusAdapter	An abstract adapter class for receiving keyboard focus events.
FocusEvent	A low-level event which indicates that a Component has gained or lost the input focus.
HierarchyBoundsAdapter	An abstract adapter class for receiving ancestor moved and resized events.
HierarchyEvent	An event which indicates a change to the Component hierarchy to which Component belongs.
InputEvent	The root event class for all component-level input events.
InputMethodEvent	Input method events contain information about text that is being composed using an input method.
InvocationEvent	An event which executes the run() method on a Runnable when dispatched by the AWT event dispatcher thread.
ItemEvent	A semantic event which indicates that an item was selected or deselected.
KeyAdapter	An abstract adapter class for receiving keyboard events.
KeyEvent	An event which indicates that a keystroke occurred in a component.
MouseAdapter	An abstract adapter class for receiving mouse events.
MouseEvent	An event which indicates that a mouse action occurred in a component.
MouseMotionAdapter	An abstract adapter class for receiving mouse motion events.
MouseWheelEvent	An event which indicates that the mouse wheel was rotated in a component.
PaintEvent	The component-level paint event.
TextEvent	A semantic event which indicates that an object's text changed.

WindowAdapter	An abstract adapter class for receiving window events.
WindowEvent	A low-level event that indicates that a window has changed its status.



At the root of Java event class hierarchy is EventObject which is in java.util. It is a super class for all events.

Question: What is Event Handling?

Answer:

Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism has the code which is known as event handler that is executed when an event occurs.

The Delegation Event Model has the following key participants namely:

Source - The source is an object on which event occurs. Source is responsible for providing information of the occurred event to its handler. Java provide as with classes for source object.

Listener - It is also known as event handler. Listener is responsible for generating response to an event. From java implementation point of view the listener is also an object. Listener waits until it receives an event. Once the event is received , the listener process the event an then returns.

Steps involved in event handling

The User clicks the button and the event is generated.

Now the object of concerned event class is created automatically and information about the source and the event get populated with in same object.

Event object is forwarded to the method of registered listener class. the method is now get executed and returns.

Event Handling Example

Create the following java program using any editor of your choice in say
D:/ > AWT > com > tutorialspoint > gui >

AwtControlDemo.java

```
package com.tutorialspoint.gui;
```

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

public class AwtControlDemo {

    private Frame mainFrame;
    private Label headerLabel;
    private Label statusLabel;
    private Panel controlPanel;

    public AwtControlDemo(){
        prepareGUI();
    }
    public static void main(String[] args){
        AwtControlDemo awtControlDemo = new
AwtControlDemo();
        awtControlDemo.showEventDemo();
    }
    private void prepareGUI(){
        mainFrame = new Frame("Java AWT Examples");
        mainFrame.setSize(400,400);
        mainFrame.setLayout(new GridLayout(3, 1));
        mainFrame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent windowEvent){
                System.exit(0);
            }
        });
        headerLabel = new Label();
        headerLabel.setAlignment(Label.CENTER);
        statusLabel = new Label();
        statusLabel.setAlignment(Label.CENTER);
        statusLabel.setSize(350,100);
        controlPanel = new Panel();
        controlPanel.setLayout(new FlowLayout());
        mainFrame.add(headerLabel);
        mainFrame.add(controlPanel);
        mainFrame.add(statusLabel);
        mainFrame.setVisible(true);
    }

    private void showEventDemo(){
        headerLabel.setText("Control in action: Button");
        Button okButton = new Button("OK");
        Button submitButton = new Button("Submit");
        Button cancelButton = new Button("Cancel");

        okButton.setActionCommand("OK");
        submitButton.setActionCommand("Submit");
```

```
cancelButton.setActionCommand("Cancel");

okButton.addActionListener(new ButtonClickListener());
submitButton.addActionListener(new ButtonClickListener());
cancelButton.addActionListener(new ButtonClickListener());
controlPanel.add(okButton);
controlPanel.add(submitButton);
controlPanel.add(cancelButton);
mainFrame.setVisible(true);
}

private class ButtonClickListener implements ActionListener{
    public void actionPerformed(ActionEvent e) {
        String command = e.getActionCommand();
        if( command.equals( "OK" )) {
            statusLabel.setText("Ok Button clicked.");
        }
        else if( command.equals( "Submit" ) ) {
            statusLabel.setText("Submit Button clicked.");
        }
        else {
            statusLabel.setText("Cancel Button clicked.");
        }
    }
}
```

Compile the program using command prompt. Go to **D:/ > AWT** and type the following command.

D:\AWT>javac com\tutorialspoint\gui\AwtControlDemo.java

If no error comes that means compilation is successful. Run the program using following command.

D:\AWT>java com.tutorialspoint.gui.AwtControlDemo

Verify the following output



Question: Describe how to handle MouseEvent and KeyEvent in Java? Exam-2015

Answer:

Handling Mouse Events

Mouse events are one of the two most frequent kinds of events handled by an application (the other kind being, of course, key events). Mouse clicks—which involve a user pressing and then releasing a mouse button—generally indicate selection,

but what the selection means is left up to the object responding to the event. For example, a mouse click could tell the responding object to alter its appearance and then send an action message. Mouse drags generally indicate that the receiving view should move itself or a drawn object within its bounds. The following sections describe how you might handle mouse-down, mouse-up, and mouse-drag events.

Overview of Mouse Events

Before going into the “how to” of mouse-event handling,

Table 4-1 Type constants and methods related to left-button mouse events

Action	Event type (left mouse button)	Mouse-event method invoked (left mouse button)
Press down the button	<u>NSLeftMouseDown</u>	<u>mouseDown:</u>
Move the mouse while pressing the button	<u>NSLeftMouseDragged</u>	<u>mouseDragged:</u>
Release the button	<u>NSLeftMouseUp</u>	<u>mouseUp:</u>
Move the mouse without pressing any button	<u>NSMouseMoved</u>	<u>mouseMoved:</u>

Mouse Event Handling in a Frame Window Example

```
/*
    Mouse Event Handling in a Frame Window Example
    This java example shows how to handle mouse events in a Frame window
    using MouseListener.
*/
```

```
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
```

```
/*
```

* To create a stand alone window, class should be extended from
* Frame and not from Applet class.
*/

```
public class HandleMouseListenerInWindowExample extends Frame implements  
MouseListener{
```

```
    int x=0, y=0;  
    String strEvent = "";
```

```
    HandleMouseListenerInWindowExample(String title){
```

```
        //call superclass constructor with window title  
        super(title);
```

```
        //add window listener  
        addWindowListener(new MyWindowAdapter(this));
```

```
        //add mouse listener  
        addMouseListener(this);
```

```
        //set window size  
        setSize(300,300);
```

```
        //show the window  
        setVisible(true);
```

```
    }
```

```
    public void mouseClicked(MouseEvent e) {
```

```
        strEvent = "MouseClicked";  
        x = e.getX();  
        y = e.getY();  
        repaint();
```

```
    }
```

```
    public void mousePressed(MouseEvent e) {
```

```
        strEvent = "MousePressed";  
        x = e.getX();  
        y = e.getY();  
        repaint();
```

```
    }
```

```
    public void mouseReleased(MouseEvent e) {
```

```
        strEvent = "MouseReleased";  
        x = e.getX();  
        y = e.getY();  
        repaint();
```

```
}

public void mouseEntered(MouseEvent e) {
    strEvent = "MouseEntered";
    x = e.getX();
    y = e.getY();
    repaint();
}

public void mouseExited(MouseEvent e) {
    strEvent = "MouseExited";
    x = e.getX();
    y = e.getY();
    repaint();
}

public void paint(Graphics g){
    g.drawString(strEvent + " at " + x + "," + y, 50,50);
}

public static void main(String[] args) {

    HandleMouseListenerInWindowExample myWindow =
        new HandleMouseListenerInWindowExample("Window With Mouse
Events Example");
}

}

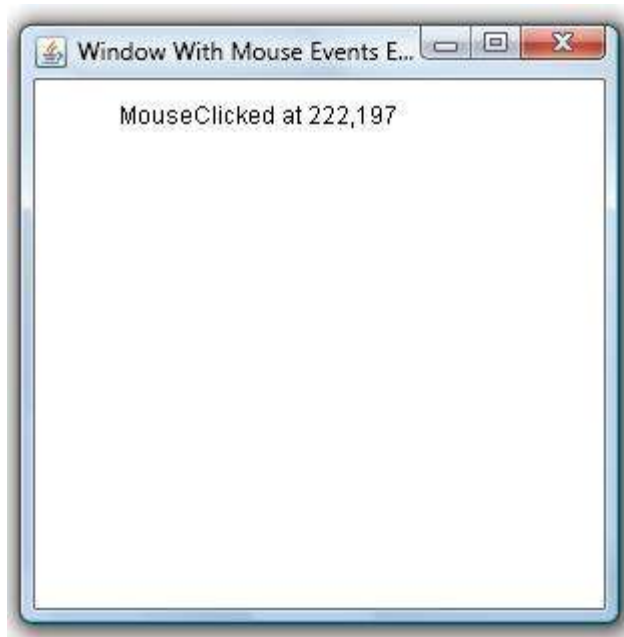
class MyWindowAdapter extends WindowAdapter{

    HandleMouseListenerInWindowExample myWindow = null;

    MyWindowAdapter(HandleMouseListenerInWindowExample myWindow){
        this.myWindow = myWindow;
    }

    public void windowClosing(WindowEvent we){
        myWindow.setVisible(false);
    }
}
```

Example Output

**KeyEvent:**

Key events indicate when the user is typing at the keyboard. Specifically, key events are fired by the component with the keyboard focus when the user presses or releases keyboard keys.

Notifications are sent about two basic kinds of key events:

- The typing of a Unicode character
- The pressing or releasing of a key on the keyboard

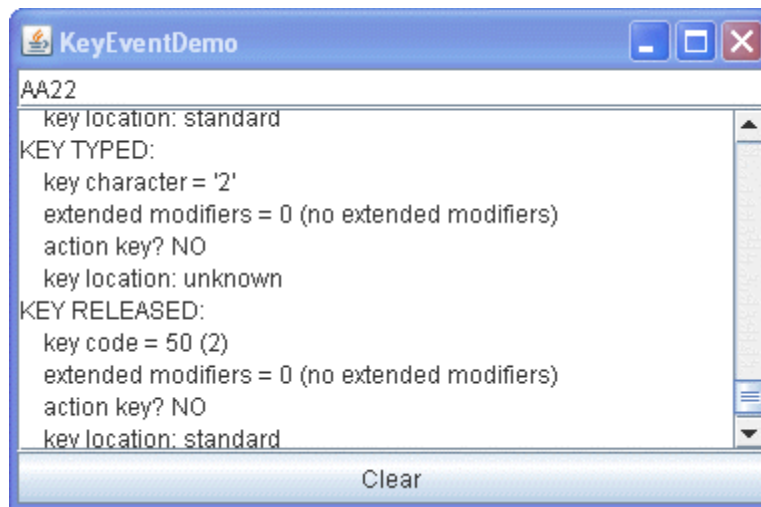
The first kind of event is called a key-typed event. The second kind is either a key-pressed or key-released event.

In general, you react to only key-typed events unless you need to know when the user presses keys that do not correspond to characters. For example, to know when the user types a Unicode character "€" whether by pressing one key such as 'a' or by pressing several keys in sequence "€" you handle key-typed events. On the other hand, to know when the user presses the F1 key, or whether the user pressed the '3' key on the number pad, you handle key-pressed events.

To make a component get the keyboard focus, follow these steps:

1. Make sure the component's `isFocusable` method returns `true`. This state allows the component to receive the focus. For example, you can enable keyboard focus for a `JLabel` component by calling the `setFocusable(true)` method on the label.
2. Make sure the component requests the focus when appropriate. For custom components, implement a mouse listener that calls the `requestFocusInWindow` method when the component is clicked.

The following example demonstrates key events. It consists of a text field that you can type into, followed by a text area that displays a message every time the text field fires a key event. A button at the bottom of the window lets you clear both the text field and text area.



Important Question



1. Define runtime exception and IO exception with example **Exam-2015**
2. In detail explain the try-catch finally blocks of the exception handling. **Exam-2015**
3. With necessary code snippet, explain the difference between the keywords and throws. **Exam-2015**
4. What do you understand by exception handling? Explain with suitable codes?. **Exam-2014**
5. Describe the five keywords that manage Java exception handling. **Exam-2013**

Exception:

An exception is an indication of a problem that occurs during a program's execution. The name "exception" implies that the problem occurs infrequently if the "rule" is that a statement normally executes correctly, and then the "exception to the rule" is that a problem occurs.

Question: What do you understand by exception handling? Explain with suitable codes?. Exam-2014

Exception Handling:

Exception handling enables programmers to create applications that can resolve (or handle) exceptions.

In many cases, handling an exception allows a program to continue executing as if no problem had been encountered. A more severe problem could prevent a program from continuing normal execution, instead requiring it to notify the user of the problem before terminating in a controlled manner

Java exception handling is managed via five keywords: try, catch, throw, throws, and finally. Program that we want to monitor for exception are written within the try block. If an exception occurs within the try block, then it is thrown.

Example:

This is the general form of an exception-handling block:

```
try {  
    // block of code to monitor for errors  
}  
catch (ExceptionType1 exOb) {  
    // exception handler for ExceptionType1  
}  
catch (ExceptionType2 exOb) {  
    // exception handler for ExceptionType2  
}  
// ...  
finally {  
    // block of code to be executed before try block ends  
}
```

Here, ExceptionType is the type of exception that has occurred.

Example of ArithmeticException.

InputMismatchException

```
import java.util.Scanner;
```

```
public class DivideByZeroNoExceptionHandling  
{  
    // demonstrates throwing an exception when a divide-by-zero occurs  
    public static int quotient( int numerator, int denominator )  
    {  
        return numerator / denominator; // possible division by zero  
    } // end method quotient  
  
    public static void main( String args[] )  
    {
```

```

Scanner scanner = new Scanner( System.in ); // scanner for input
System.out.print( "Please enter an integer numerator: " );
int numerator = scanner.nextInt();
System.out.print( "Please enter an integer denominator: " );
int denominator = scanner.nextInt();
int result = quotient( numerator, denominator );
System.out.printf(
    "\nResult: %d / %d = %d\n", numerator, denominator, result );
} // end main
} // end class DivideByZeroNoExceptionHandling

```

```

Please enter an integer numerator: 100
Please enter an integer denominator: 7

Result: 100 / 7 = 14

```

```

Please enter an integer numerator: 100
Please enter an integer denominator: 0
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at DivideByZeroNoExceptionHandling.quotient(DivideByZeroNoException-
Handling.java:10)
    at DivideByZeroNoExceptionHandling.main(DivideByZeroNoExceptionHan-
dling.java:22)

```

```

Please enter an integer numerator: 100
Please e
nter an integer denominator: hello
Exception in thread "main" java.util.InputMismatchException
    at java.util.Scanner.throwFor(Unknown Source)
    at java.util.Scanner.next(Unknown Source)
    at java.util.Scanner.nextInt(Unknown Source)
    at java.util.Scanner.nextInt(Unknown Source)
    at DivideByZeroNoExceptionHandling.main(DivideByZeroNoExceptionHan-
dling.java:20)

```

Figure 13.2. Handling ArithmeticExceptions and InputMismatchExceptions.

(This item is displayed on pages 643 - 644 in the print version)

```

// Fig. 13.2: DivideByZeroWithExceptionHandling.java
// An exception-handling example that checks for divide-by-zero.
import java.util.InputMismatchException;
import java.util.Scanner;

```

```

public class DivideByZeroWithExceptionHandling
{

```



```
// demonstrates throwing an exception when a divide-by-zero occurs
public static int quotient( int numerator, int denominator )
    throws ArithmeticException
{
    return numerator / denominator; // possible division by zero
} // end method quotient

public static void main( String args[] )
{
    Scanner scanner = new Scanner( System.in ); // scanner for input
    boolean continueLoop = true; // determines if more input is needed

do
{
    try // read two numbers and calculate quotient
    {
        System.out.print( "Please enter an integer numerator: " );
        int numerator = scanner.nextInt();
        System.out.print( "Please enter an integer denominator: " );
        int denominator = scanner.nextInt();
        int result = quotient( numerator, denominator );
        System.out.printf( "\nResult: %d / %d = %d\n", numerator,
            denominator, result );
        continueLoop = false; // input successful; end looping
    } // end try
    catch ( InputMismatchException inputMismatchException )
    {
        System.err.printf( "\nException: %s\n",
            inputMismatchException );
        scanner.nextLine(); // discard input so user can try again
        System.out.println(
            "You must enter integers. Please try again.\n" );
    } // end catch
    catch ( ArithmeticException arithmeticException )
    {
        System.err.printf( "\nException: %s\n", arithmeticException );
        System.out.println(
            "Zero is an invalid denominator. Please try again.\n" );
    } // end catch
} while ( continueLoop ); // end do...while
} // end main
} // end class DivideByZeroWithExceptionHandling
```

```
Please enter an integer numerator: 100
Please enter an integer denominator: 7

Result: 100 / 7 = 14
```

```
Please enter an integer numerator: 100
Please enter an integer denominator: 0

Exception: java.lang.ArithmeticException: / by zero
Zero is an invalid denominator. Please try again.

Please enter an integer numerator: 100
Please enter an integer denominator: 7

Result: 100 / 7 = 14
```

Using the throws Clause

A **throws** clause specifies the exceptions the method **throws**. This clause appears after the method's parameter list and before the method's body. The clause contains a comma-separated list of the exceptions that the method will throw if a problem occurs. Such exceptions may be thrown by statements in the method's body or by methods called in the body. A method can throw exceptions of the classes listed in its **throws** clause or of their subclasses. We have added the **throws** clause to this application to indicate to the rest of the program that this method may throw an **ArithmeticException**. Clients of method **quotient** are thus informed that the method may throw an **ArithmeticException** and that the exception should be caught.

```
Please enter an integer numerator: 100
Please enter an integer denominator: hello

Exception: java.util.InputMismatchException
You must enter integers. Please try again.

Please enter an integer numerator: 100
Please enter an integer denominator: 7

Result: 100 / 7 = 14
```

Question: In detail explain the try-catch finally blocks of the exception handling. Exam-2015

Answer:

The try Block

The try block encases any statements that might cause an exception to occur.

For example,

if you are reading data from a file using the FileReader class its expected that you handle the IOExceptions associated with using a FileReader object (e.g, FileNotFoundException, IOException). To ensure this happens you can place the statements that deal with creating and using the FileReader object inside a try block:

```
public static void main(String[] args){  
    FileReader fileInput = null;
```

```
        try  
        {  
            //Open the input file  
            fileInput = new FileReader("Untitled.txt");  
        }  
    }
```

However, the code is incomplete because in order for the exception to be handled we need a place for it to be caught. This happens in the catch block .

The catch Block

The catch block(s) provide a place to handle the exception thrown by the statements within a try block. The catch block is defined directly after the try block.

It must specify the type of exception it is handling.

For example,

the FileReader object defined in the code above is capable of throwing a FileNotFoundException or an IOException. We can specify two catch blocks to handle both of those exceptions:

```
public static void main(String[] args){  
    FileReader fileInput = null;
```

```
        try  
        {  
            //Open the input file  
            fileInput = new FileReader("Untitled.txt");  
        }  
        catch(FileNotFoundException ex)  
        {  
            //handle the FileNotFoundException  
        }  
        catch(IOException ex)  
        {  
            //handle the IOException  
        }  
    }
```

In the FileNotFoundException catch block we could place code to ask the user to find the file for us and then try to read the file again. In the IOException catch block we might just pass on the I/O error to the user and ask them to try something else. Either way, we have provided a way for the program to catch an exception and handle it in a controlled manner.

The finally Block

The statements in the finally block are always executed. This is useful to clean up resources in the event of the try block executing without an exception and in the cases when there is an exception. In both eventualities, we can close the file we have been using.






The finally block appears directly after the last catch block:

```
public static void main(String[] args){
    FileReader fileInput = null;

    try
    {
        //Open the input file
        fileInput = new FileReader("Untitled.txt");
    }
    catch(FileNotFoundException | IOException ex)
    {
        //handle both exceptions
    }
    finally
    {
        //We must remember to close streams
        //Check to see if they are null in case there was an
        //IO error and they are never initialised
        if (fileInput != null)
        {
            fileInput.close();
        }
    }
}
```

Question: Describe the five keywords that manage Java exception handling. Exam-2013

Answer:

-  try
-  catch
-  finally
-  throw
-  throws

Although we have covered every keyword individually, let us summarize each keyword with few lines and finally one example covering each keyword in a single program

Throw clause:

Sometimes, programmer can also **throw/raise exception explicitly at runtime** on the basis of some business condition

To **raise such exception explicitly** during program execution, we need to use **throw** keyword

Syntax: throw instanceofThrowableType;

Generally, **throw keyword is used to throw user-defined exception_or custom exception**

Although, it is perfectly valid to throw pre-defined exception or already defined exception in Java like IOException, NullPointerException, ArithmeticException, InterruptedException, ArrayIndexOutOfBoundsException, etc.

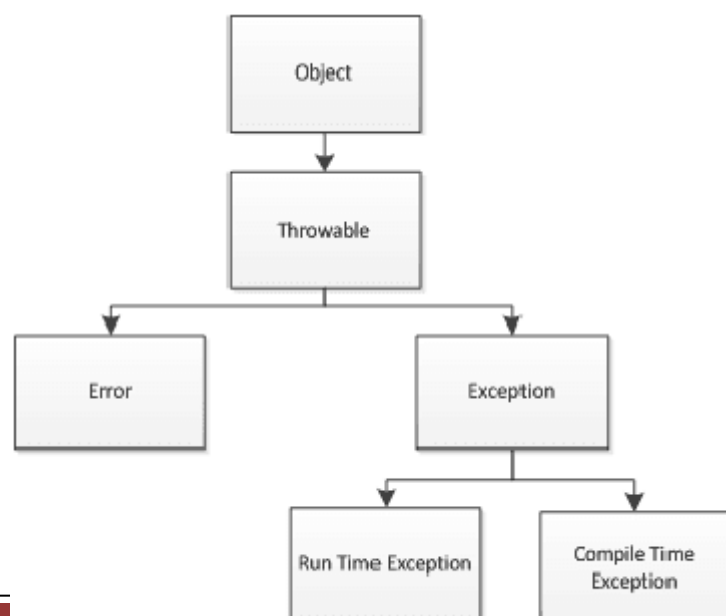
Pseudo code:

```
try {  
  
    // some valid Java statements  
    throw new RuntimeException();  
}  
catch(Throwable th) {  
  
    // handle exception here  
    // or re-throw caught exception  
}
```

throws keyword or throws clause:

- 📖 **throws keyword** is used to **declare the exception** that might raise during program execution
- 📖 whenever exception might thrown from program, then programmer doesn't necessarily need to handle that exception using **try-catch block** instead simply **declare that exception** using throws clause next to method signature
- 📖 But this **forces or tells** the caller method to handle that exception; but again caller can handle that exception using **try-catch block** or **re-declare those exception** with **throws clause**
- 📖 **Note:** use of throws clause doesn't necessarily mean that program will terminate normally rather it is the information to the caller to handle for normal termination
- 📖 Any **number of exceptions** can be specified using throws clause, but they are all need to be separated by commas (,)
- 📖 **throws clause** is applicable for **methods & constructor** but strictly not applicable to classes
- 📖 It is mainly used for [checked exception](#), as [unchecked exception](#) by default propagated back to the caller (i.e.; up in the runtime stack)
- 📖 **Note:** It is highly recommended to use try-catch for exception handling instead of throwing exception using throws clause

The exception is object created at the time of exceptional/error condition which will be thrown from the program and halt normal execution of the program. Java exceptions object hierarchy is as below:



Java's exceptions can be categorized into two types:

1. Checked exceptions
2. Unchecked exceptions

checked exceptions are subject to the catch or specify a requirement, which means they require catching or declaration. This requirement is optional for unchecked exceptions. Code that uses a checked exception will not compile if the catch or specify rule is not followed.

Unchecked exceptions come in two types:

- I. Errors
- II. Runtime exceptions

Checked Exceptions

Checked exceptions are the type that programmers should anticipate and from which programs should be able to recover. All Java exceptions are checked exceptions except those of the Error and RuntimeException classes and their subclasses.

These could include subclasses of FileNotFoundException, UnknownHostException, etc.

Popular Checked Exceptions:

Name	Description
IOException	While using file input/output stream related exception
SQLException.	While executing queries on database related to SQL syntax
DataAccessException	Exception related to accessing data/database
ClassNotFoundException	Thrown when the JVM can't find a class it needs, because of a command-line error, a classpath issue, or a missing .class file
InstantiationException	Attempt to create an object of an abstract class or interface.

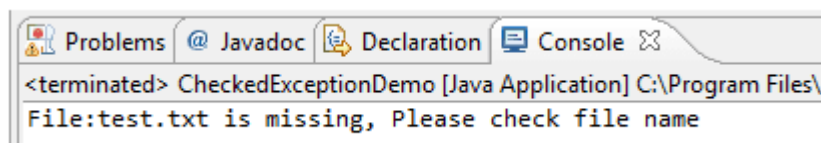
Below example program of reading, file shows how checked exception should be handled.

```
public String readFile(String filename){
    FileInputStream fin;
    int i;
    String s="";
    fin = new FileInputStream(filename);
    // read characters until EOF is en
    do {
        i = fin.read();
        if(i != -1) s =(char) i+"";
    } while(i != -1);
    fin.close();
    return s;
}
```

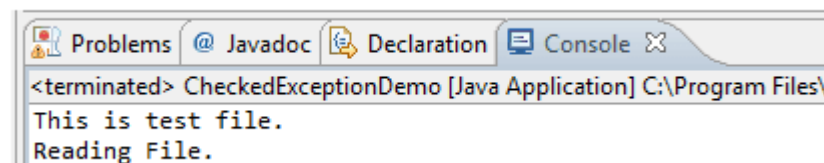


```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
public class CheckedExceptionDemo {
    public static void main(String[] args) {
        //Below line calls readFile method and prints content of it
        String filename="test.txt";
        try {
            String fileContent = new CheckedExceptionDemo().readFile(filename);
            System.out.println(fileContent);
        } catch (FileNotFoundException e) {
            System.out.println("File:" + filename + " is missing, Please check file name"
);
        } catch (IOException e) {
            System.out.println("File is not having permission to read, please check the
permission");
        }
    }
    public String readFile(String filename)throws FileNotFoundException, IOExce
ption{
        FileInputStream fin;
        int i;
        String s="";
        fin = new FileInputStream(filename);
        // read characters until EOF is encountered
        do {
            i = fin.read();
            if(i != -1) s=s+(char) i+"";
        } while(i != -1);
        fin.close();
        return s;
    }
}
```

Output: If test.txt is not found:



Running the program after creating test.txt file inside project root folder



Unchecked Exceptions

Unchecked exceptions inherit from the Error class or the RuntimeException class. Many programmers feel that you should not handle these exceptions in your programs

because they represent the type of errors from which programs cannot reasonably be expected to recover while the program is running.

When an unchecked exception is thrown, it is usually caused by a misuse of code - passing a null or otherwise incorrect argument.

Popular Unchecked Exceptions:

Name	Description
NullPointerException	Thrown when attempting to access an object with a reference variable whose current value is null
ArrayIndexOutOfBoundsException	Thrown when attempting to access an array with an invalid index value (either negative or beyond the length of the array)
IllegalArgumentException	Thrown when a method receives an argument formatted differently than the method expects.
IllegalStateException	Thrown when the state of the environment doesn't match the operation being attempted, e.g., using a Scanner that's been closed.
NumberFormatException	Thrown when a method that converts a String to a number receives a String that it cannot convert.
ArithmeticException	Arithmetic error, such as divide-by-zero.

Question: Define runtime exception and IO exception with example Exam-2015

RuntimeException

An unchecked exception is an exception that occurs at the time of execution. These are also called as Runtime.

I also use Runtime Exception for the default case of a switch statement if there is no better way of handling it.

```
public class RuntimeException  
    extends Exception
```

RuntimeException is the superclass of those exceptions that can be thrown during the normal operation of the Java Virtual Machine.





Example;

IOException:

In general, I/O means Input or Output. Those methods throw the IOException whenever an input or output operation is failed or interpreted.

Note that this won't be thrown for reading or writing to memory as Java will be handling it automatically. Here are some cases which result in IOException.

IO Exception may occur –

-  when you read some file from hard disk,
-  You are trying to read/write a file and don't have permission,
-  You were writing a file and disk space is not available anymore, etc.
-  so consider the simple scenario, you are trying to read a file from hard disk - so you provided exact path of that file, but the file doesn't exist! so java would throw IO Exception.

Interface Summary

S.N.	Interface & Description
1	CharConversionException This is a base class for character conversion exceptions.
2	EOFException These are signals that an end of file or end of stream has been reached unexpectedly during input.
3	FileNotFoundException These are the signals that an attempt to open the file denoted by a specified pathname has failed.
4	InterruptedIOException This is signals that an I/O operation has been interrupted.
5	InvalidClassException This is thrown when the Serialization runtime detects one of the following problems with a Class.
6	InvalidObjectException This indicates that one or more deserialized objects failed validation tests.
7	IOException These are the signals that an I/O exception of some sort has occurred.
8	NotActiveException This is thrown when serialization or deserialization is not active.
9	NotSerializableException This is thrown when an instance is required to have a Serializable interface.
10	ObjectStreamException This is a superclass of all exceptions specific to Object Stream classes.
11	OptionalDataException This is an exception indicating the failure of an object read operation due to unread primitive data, or the end of data belonging to a serialized object in the stream.
12	StreamCorruptedException This is thrown when control information that was read from an object stream violates internal consistency checks.

13 SyncFailedException

These are the signals that a sync operation has failed.

14 UnsupportedEncodingException

This character encoding is not supported.

RuntimeException vs Checked Exception in Java

Java Exceptions are divided in two categories RuntimeException also known as unchecked Exception and checked Exception.

Checked Exception	RuntimeException
, It is mandatory to provide try catch or try finally block to handle checked Exception and failure to do so will result in compile time error,	while in case of RuntimeException this is not mandatory
mandatory exception handling is not requirement for them	Any Exception which is subclass of RuntimeException are called unchecked
. . Popular example of checked Exceptions are ClassNotFoundException and IOException and that's the reason you need to provide a try catch finally block while performing file operations in Java as many of them throws IOException	Some of the most common Exception like NullPointerException, ArrayIndexOutOfBoundsException are unchecked and they are descended from java.lang.RuntimeException

Question: With necessary code snippet, explain the difference between the keywords and throws.
Exam-2015

Code Snipt:

Snippet is a programming term for a small region of re-usable source code, machine code, or text. Ordinarily, these are formally defined operative units to incorporate into larger programming modules.

In programming practice, "snippet" refers narrowly to a portion of source code that is literally included by an editor program into a file, and is a form of copy and paste programming. This concrete inclusion is in contrast to abstraction methods, such as functions or macros, which are abstraction within the language. Snippets are thus primarily used when these abstractions are not available or not desired, such as in languages that lack abstraction, or for clarity and absence of overhead.

Snippets are similar to having static preprocessing included in the editor, and do not require support by a compiler. On the flip side, this means that snippets cannot be invariably modified after the fact, and thus is vulnerable to all of the problems of copy and paste programming. For this reason snippets are primarily used for simple sections of code (with little logic), or for boilerplate, such as copyright notices, function prototypes, common control structures, or standard library imports.

Difference between the throw keywords and throws:

There are many differences between throw and throws keywords. A list of differences between throw and throws are given below:

No.	throw	throws
1)	Java throw keyword is used to explicitly throw an exception.	Java throws keyword is used to declare an exception.
2)	Checked exception cannot be propagated using throw only.	Checked exception can be propagated with throws.
3)	Throw is followed by an instance.	Throws is followed by class.
4)	Throw is used within the method.	Throws is used with the method signature.
5)	You cannot throw multiple exceptions.	You can declare multiple exceptions e.g. public void method()throws IOException,SQLException.

Throw vs Throws in java

1. **Throws clause** is used to declare an exception, which means it works similar to the try-catch block. On the other hand **throw** keyword is used to throw an exception explicitly.

2. If we see syntax wise than **throw** is followed by an instance of Exception class and **throws** is followed by exception class names.
For example:

About the Authors

Abu Saleh Musa Miah(abid) was born in Rangpur,Bangladesh in 1992. He received the B.Sc. Engineering degree with Honours in Computer science and Engineering in 2014 with FIRST merit Position Engineering first batch from University of Rajshahi ,Bangladesh. He also completed research on the field **COMPUTER VISION** specifically Moving Object Detection with BoofCV tools(java). Currently he is working towards the M.Sc.Engineering degree in the Department of Computer Science and Engineering University of Rajshahi, Bangladesh.



His research interests include **Brain Computer Interfacing**. Sparse signal recovery/compressed sensing, blind source separation, neuroimaging and computational and cognitive, now he is a Research Student, Centre for Research & Innovation (CRI Unit-I), Signal Processing & Computational Neuroscience Laboratory Dept. of Computer Science & Engineering (CSE) University of Rajshahi

Email: abusalehcse.ru@gmail.com;

<https://www.facebook.com/abusalehmusa.miah>