



## Introduction to Java



# Introduction to JAVA

What is JAVA?

- JAVA is a Pure Object Oriented Programming Language developed by SunMicroSystems.
- Java is a Platform (JRE and API).

**Platform:** It's any hardware or software environment in which a program runs.

# Introduction to JAVA (contd.)



Few  
Languages  
before  
JAVA?

- Machine Code
- Assembly Language
- C
- Smalltalk
- C++
- Java

# Introduction to JAVA (contd.)

- History of JAVA
- Java was developed by James Gosling, Patrick Naughton and Mike Sheridan.
- It was initiated in 1991 and was called as Green.
- It was initially developed in 1993 and was initially named Oak.
- Finally in 1995 Java 1.0 was released.

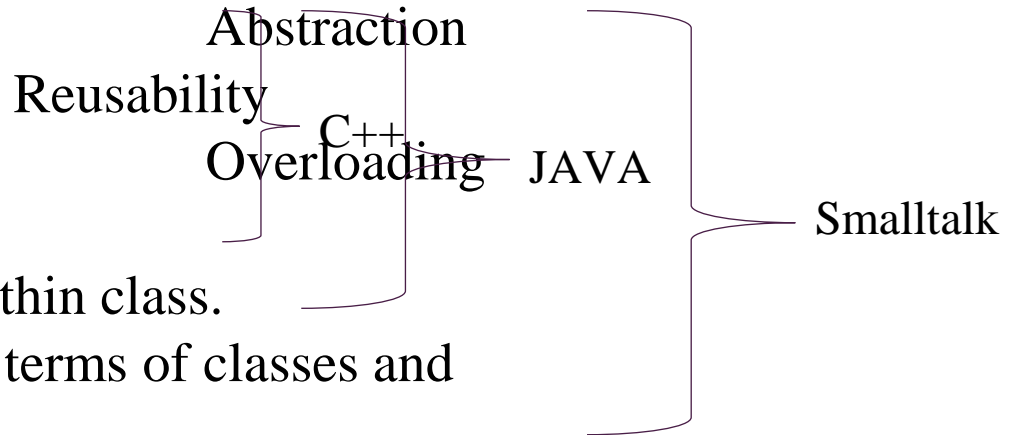
# Introduction to JAVA (contd.)

- \*7 : First project developed using Java (1993-mids)
- HotJava : browser to advertise about Java (1995)
- Netscape : gave Java it's first break (1996)

# Introduction to JAVA (contd.)

Properties of OOP Language:

- Encapsulation →
- Inheritance →
- Polymorphism →
- Overriding →



- Everything must be written within class.
- Everything must be created in terms of classes and objects.
- Every actions must be performed by method invocations.

# Different Paradigms

- Imperative Paradigm
- Functional Paradigm
- Object-Oriented Paradigm
- Procedural Paradigm
- Logic Programming Paradigm
- Event-Driven Paradigm
- Scripting Paradigm
- Concurrent and Parallel Paradigms

# Why Java ?

- Platform Independence
- Object-Oriented Programming
- Key Language Features
- The Java Virtual Machine (JVM)
- Java Standard Library (Java API)
- Robustness and Security
- Multi-threading
- Portability
- Java Community and Ecosystem
- Scalability



# Flavors of Java

- **Java Standard Edition (Java SE):**
  - Overview of Java SE
  - Core features and libraries
  - General-purpose applications
- **Java Enterprise Edition (Java EE):**
  - Introduction to Java EE
  - Enterprise-level features and APIs
  - Building web and enterprise applications
- **Java Micro Edition (Java ME):**
  - Overview of Java ME
  - Mobile and embedded systems
  - Developing applications for small devices
- **JavaFX:**
  - Introduction to JavaFX
  - Rich client application development
  - User interface and multimedia capabilities
- **Android Development:**
  - Android as a Java-based platform
  - Mobile app development for Android
  - Android-specific APIs and tools
- **Java Card:**
  - Introduction to Java Card
  - Developing smart card applications
  - Security and authentication



# Java Design Goal

- **Platform Independence:** One of the central design goals of Java was to create a language that could run on any platform without modification. To achieve this, Java uses the concept of the Java Virtual Machine (JVM), which allows Java programs to be compiled into bytecode that can run on any system with a compatible JVM. This "write once, run anywhere" philosophy was a fundamental principle in Java's design.
- **Object-Oriented:** Java was designed to be a pure object-oriented programming (OOP) language, emphasizing the use of objects, classes, and inheritance. OOP promotes modularity, reusability, and ease of maintenance, which were important considerations in the language's design.
- **Robustness:** Java aimed to be a robust language by including features like automatic memory management through garbage collection, strong type checking, and exception handling. These features help prevent common programming errors and make Java programs more reliable.
- **Security:** Security was a significant concern in Java's design. The language incorporates a robust security model that protects against various types of vulnerabilities, including buffer overflows and unauthorized access. Java applets, for instance, run in a secure sandboxed environment to prevent malicious code execution.
- **Portability:** Java's platform independence contributes to its portability. The ability to run the same Java code on different platforms reduces the need for platform-specific adaptations, making software development and distribution more efficient.
- **Multi-threading:** Java includes built-in support for multi-threading, allowing developers to write concurrent and parallel programs easily. This feature is vital for creating responsive and efficient applications.
- **Simplicity and Familiarity:** Java's syntax was designed to be easy to learn and read. It borrowed many elements from C and C++ to make it familiar to developers who were already familiar with these languages.
- **High Performance:** While Java prioritizes safety and portability, it also aimed to deliver good performance. Features like Just-In-Time (JIT) compilation help Java programs execute efficiently.
- **Rich Standard Library:** Java provides a comprehensive standard library (Java API) that includes pre-built classes and methods for various tasks, reducing the need for developers to reinvent the wheel and speeding up development.

# Role of Java Programming in Industry

- Web Development
- Enterprise Software
- Mobile App Development (Android)
- Financial Services
- Healthcare
- Telecommunications
- Gaming
- Retail and E-commerce
- Aerospace and Defense
- Education and Research
- IoT (Internet of Things)
- Big Data and Data Analysis

# Features of Java Language

- Platform Independence
- Object-Oriented
- Robustness
- Security
- Simplicity and Readability
- High Performance
- Multi-threading
- Portability
- Rich Standard Library (Java API)
- Dynamic and Extensible
- Community and Ecosystem
- Scalability
- Regular Updates

# Difference between JDK, JRE and JVM

## JDK (Java Development Kit):

- JDK is a software package that includes tools for developing, compiling, and debugging Java applications.
- It contains the Java compiler (javac), development tools (e.g., javadoc), and libraries necessary for building Java applications.
- JDK is used by developers to write, compile, and test Java code.
- It includes the JRE, so developers can run their Java applications for testing and debugging.

## JRE (Java Runtime Environment):

- JRE is a runtime environment that allows users to run Java applications and applets.
- It includes the JVM and a set of libraries and classes needed for executing Java programs.
- JRE does not include development tools or compilers; it's meant for end-users who only need to run Java applications.

## JVM (Java Virtual Machine):

- JVM is an integral part of both the JDK and JRE.
- It is a virtualized execution environment that interprets and executes Java bytecode.
- JVM abstracts the underlying hardware and operating system, allowing Java programs to be platform-independent.
- There are different JVM implementations (e.g., Oracle HotSpot, OpenJ9) that may offer variations in performance and optimizations.

# Classes and Objects

- Definitions:
- **Class** is a blueprint or template of an object.
- E.g.: blueprint of a building
- **Object** is an instance of a class or any real world entity.
- E.g.: building

# Classes and Objects(contd.)

- Class name and file name should be same if that class is 'public'.
- We can't create more than one public class in a same file.
- '.java' file is not converted to '.class'
- '.class' file is made of every class regardless of 'main method' and 'public class'.
- We can create more than one 'main method' in a single file.
- We can execute only that class which has 'main method'.
- It's a good practice to keep the main method in the class which has 'business logic'.

# Classes and Objects(contd.)

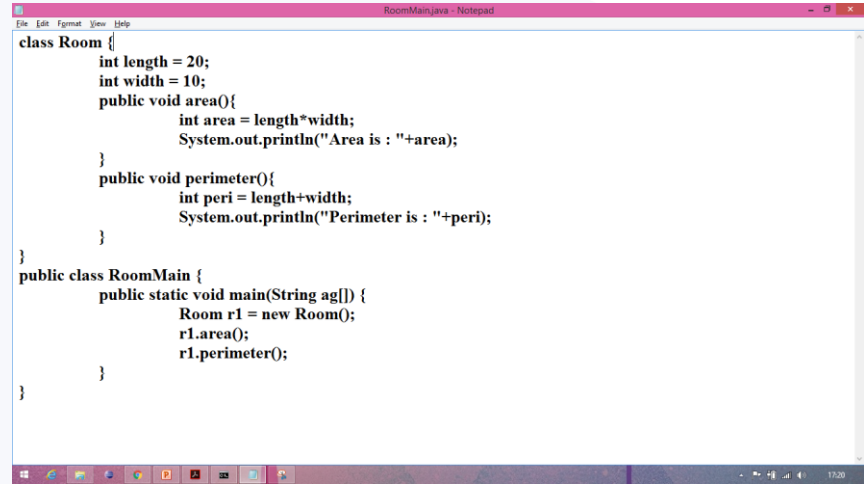
## Naming convention in Java (CamelCase)

Noun		ClassName	
Verb		methodName	
Entity		variable	



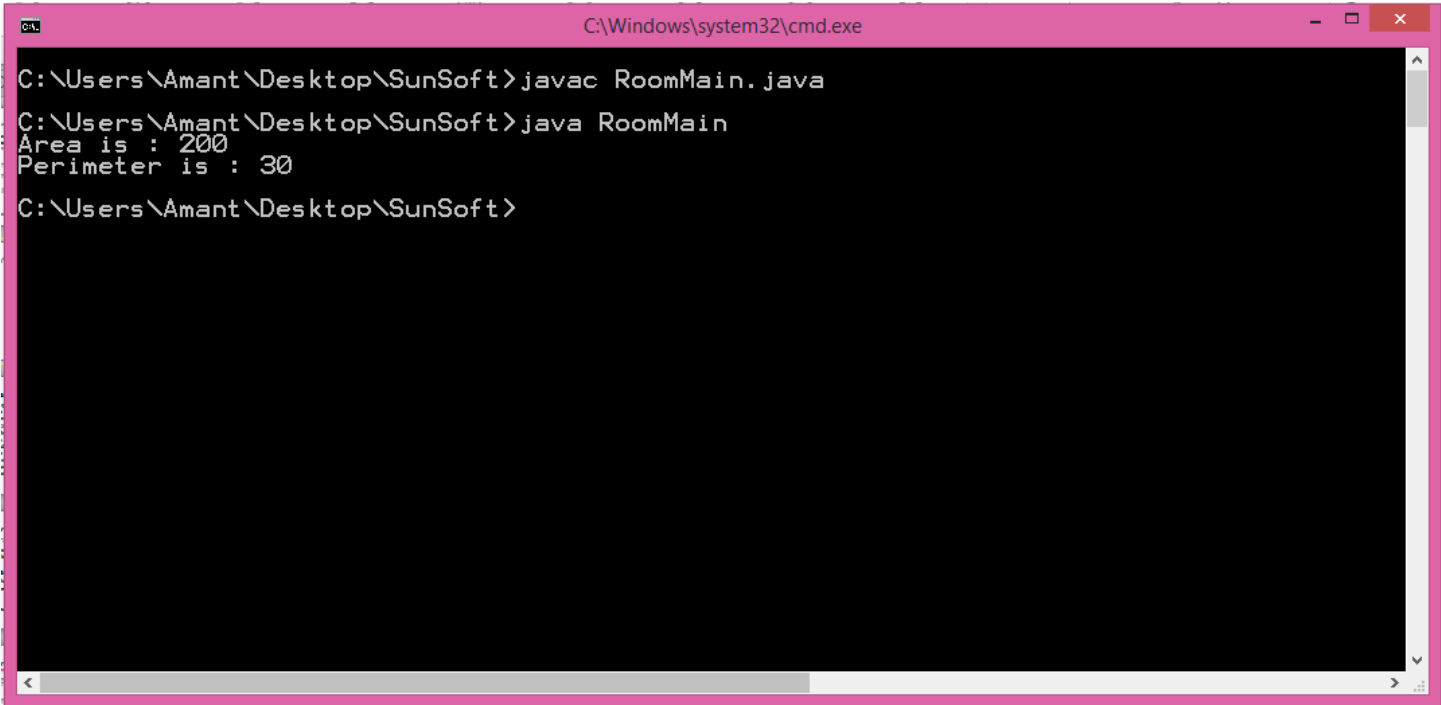
# 'new' keyword

- 'new' keyword creates the object of a class.
- 'new' keyword assigns memory when any object is created.



```
RoomMain.java - Notepad
File Edit Format View Help
class Room {
    int length = 20;
    int width = 10;
    public void area(){
        int area = length*width;
        System.out.println("Area is : "+area);
    }
    public void perimeter(){
        int peri = length+width;
        System.out.println("Perimeter is : "+peri);
    }
}
public class RoomMain {
    public static void main(String ag[]) {
        Room r1 = new Room();
        r1.area();
        r1.perimeter();
    }
}
```

## 'new' keyword (contd.)



```
C:\Windows\system32\cmd.exe

C:\Users\Amant\Desktop\SunSoft>javac RoomMain.java
C:\Users\Amant\Desktop\SunSoft>java RoomMain
Area is : 200
Perimeter is : 30
C:\Users\Amant\Desktop\SunSoft>
```

The image shows a Windows command prompt window with a pink title bar. The title bar text is 'C:\Windows\system32\cmd.exe'. The command prompt shows the following sequence of commands and output: 1. 'javac RoomMain.java' is executed. 2. 'java RoomMain' is executed, resulting in two lines of output: 'Area is : 200' and 'Perimeter is : 30'. 3. The prompt returns to 'C:\Users\Amant\Desktop\SunSoft>'. The window has standard Windows window controls (minimize, maximize, close) in the top right corner and a scrollbar on the right side.

# javac and java command

- javac is a component of JDK.
- It is used to compile the program i.e. it checks the syntactical errors and converts source code into byte code.
- java is a component of JDK as well as JRE.
- It is used to run the “.class’ file i.e. it converts byte code into machine code.

# Constructors and overloading constructors

- **Constructors are a kind of special method.**
- Constructors must have the same name as the class itself whereas methods can have any name including the class name.
- There is no need of calling constructors, unlike methods constructors are automatically called when we create object of the class.
- We can invoke methods 'n' number of times but constructors can be invoked only once i.e. during the creation of object.
- We can put any logic in constructors alike methods.
- Return type can't be used with constructors.
- If return type(s) are used with constructors then it behaves like method.
- Non-access modifiers can't be used with constructors.

# Constructors and overloading constructors (contd.)

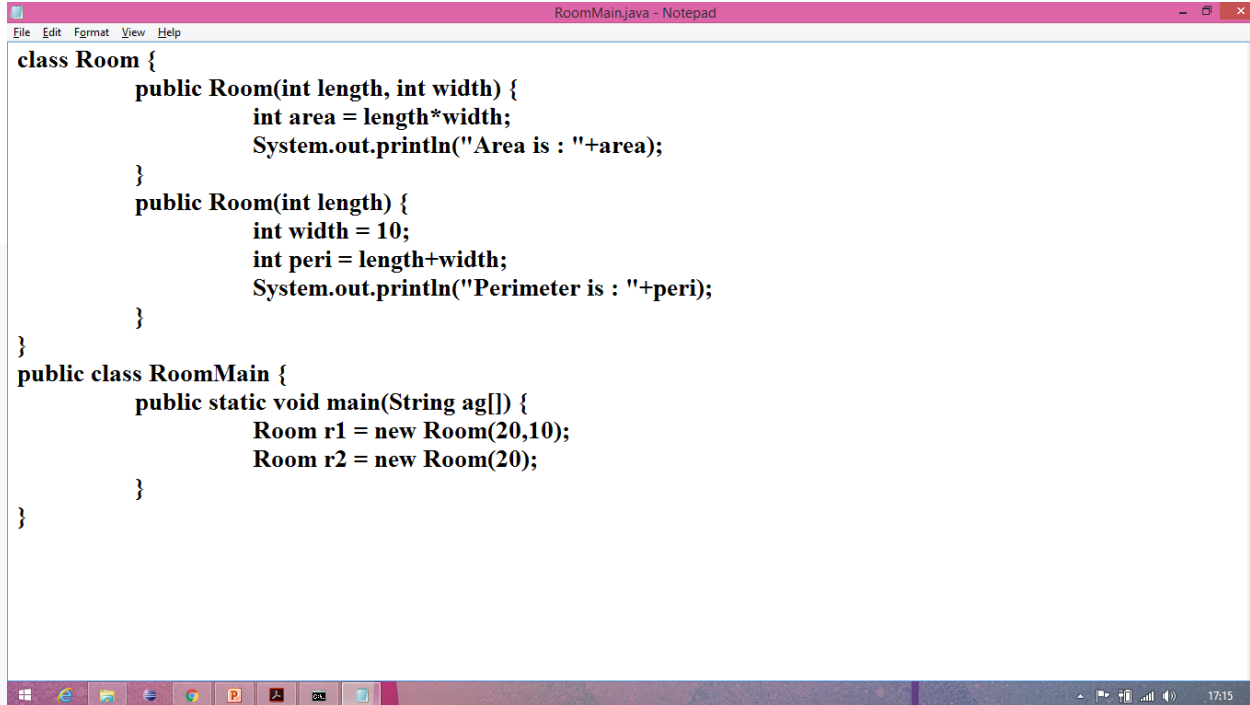
- **Default Constructor:**

- Default constructor is provided by compiler, which is always empty and if any constructor is declared explicitly then no default constructor is provided.

- **Constructor Overloading:**

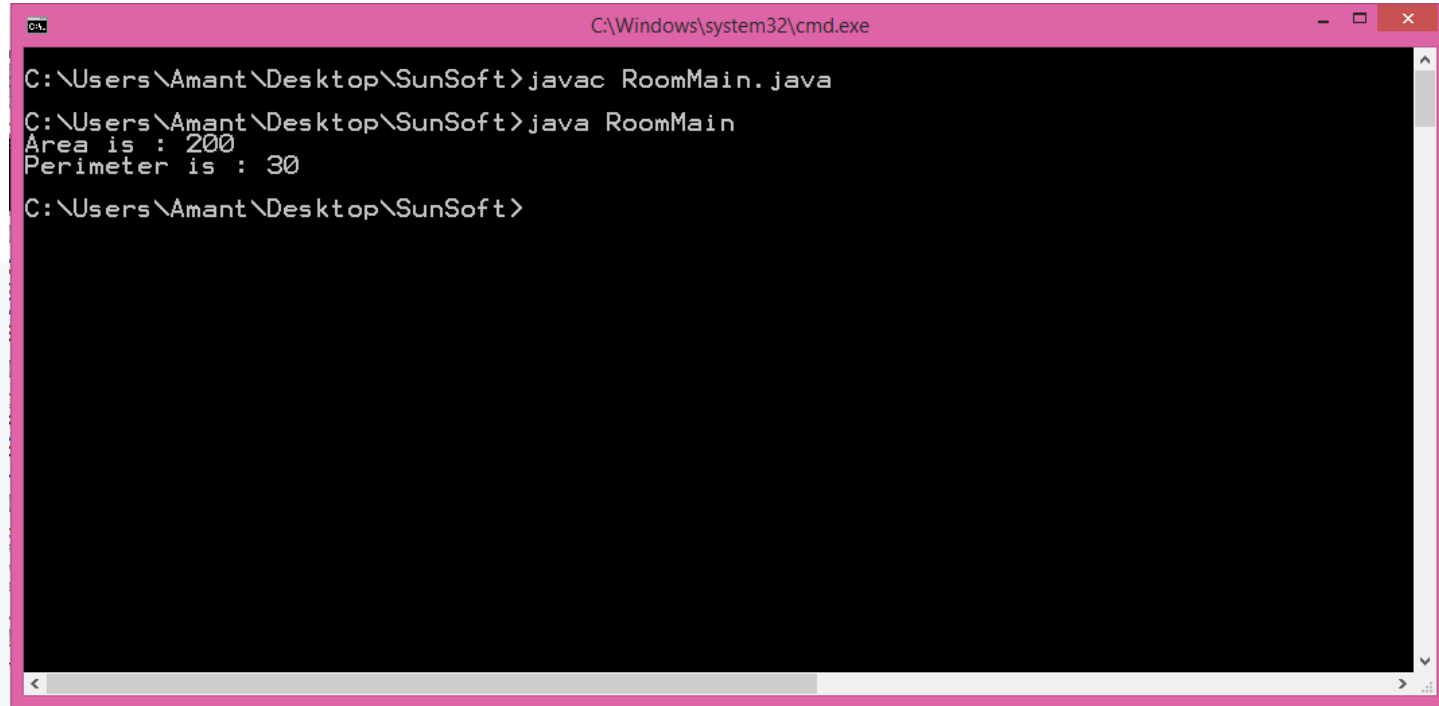
- Constructor overloading is the phenomenon of creating more than one constructor with same name but with different parameter list in the same class.
- We can't override a constructor where method overriding is possible.

# Constructors and overloading constructors (contd.)

A screenshot of a Notepad window titled "RoomMain.java - Notepad". The window contains Java code defining a "Room" class with two constructors and a "RoomMain" class with a main method. The "Room" class has a constructor that takes both length and width to calculate area, and another that takes only length to calculate perimeter. The "RoomMain" class uses these constructors to create two Room objects.

```
class Room {  
    public Room(int length, int width) {  
        int area = length*width;  
        System.out.println("Area is : "+area);  
    }  
    public Room(int length) {  
        int width = 10;  
        int peri = length+width;  
        System.out.println("Perimeter is : "+peri);  
    }  
}  
public class RoomMain {  
    public static void main(String ag[]) {  
        Room r1 = new Room(20,10);  
        Room r2 = new Room(20);  
    }  
}
```

# Constructors and overloading constructors (contd.)



```
C:\Windows\system32\cmd.exe

C:\Users\Amant\Desktop\SunSoft>javac RoomMain.java
C:\Users\Amant\Desktop\SunSoft>java RoomMain
Area is : 200
Perimeter is : 30
C:\Users\Amant\Desktop\SunSoft>
```

The screenshot shows a Windows command prompt window with a pink title bar. The title bar text is "C:\Windows\system32\cmd.exe". The command prompt shows the following sequence of commands and output:

- Command: `javac RoomMain.java`
- Command: `java RoomMain`
- Output: `Area is : 200`
- Output: `Perimeter is : 30`
- Command: `C:\Users\Amant\Desktop\SunSoft>`

# instance and static variables



## instance variables:

Instance variables are declared inside a class but outside any block like methods or constructors.

Instance variables are created when an object is created and destroyed when the object is destroyed.

Instance variables are stored in heap memory.

The scope of instance variables is throughout the class.

Instance variables can be directly invoked by calling the variable name.

Default values of instance variables:

int '0'

boolean 'false'

String 'null'



# instance and static variables (contd.)

## static variables:

Static variables are also called as 'Class variables'.

Static variables are declared inside a class with 'static' keyword, but outside any block like methods or constructor.

Static variables have only one copy throughout the class regardless of number of object created.

Static variables are stored in static memory.

It is mostly used to declare constants.

Default values of static variables are same as instance variables.

Static variables can be invoked by calling the class name.

ClassName.variable\_name



# instance and static methods

## instance methods:

- Instance methods are declared normally i.e. without static keyword.
- Instance methods belong to the object of the class.
- Instance methods can be called after creating the object of the class.
- Instance methods are stored in permanent generation of heap memory but their parameters and their local variables and value to be returned are stored in stack.
- Instance methods can be overridden.

# instance and static methods (contd.)

## Static methods:

- Static methods belongs to class.
- Static methods are declared by using static keyword.
- Just like static variables, only one copy of static method is passed throughout the class regardless of number of object created.
- Static methods are stored in permanent generation of heap memory.
- Static methods can be overloaded but cannot be overridden.
- Static methods can be invoked by calling the class name.
  - `ClassName.methodName()`
- Static methods can't access instance methods and instance variables directly but instance method can access static variables and static methods directly.

# instance and static block

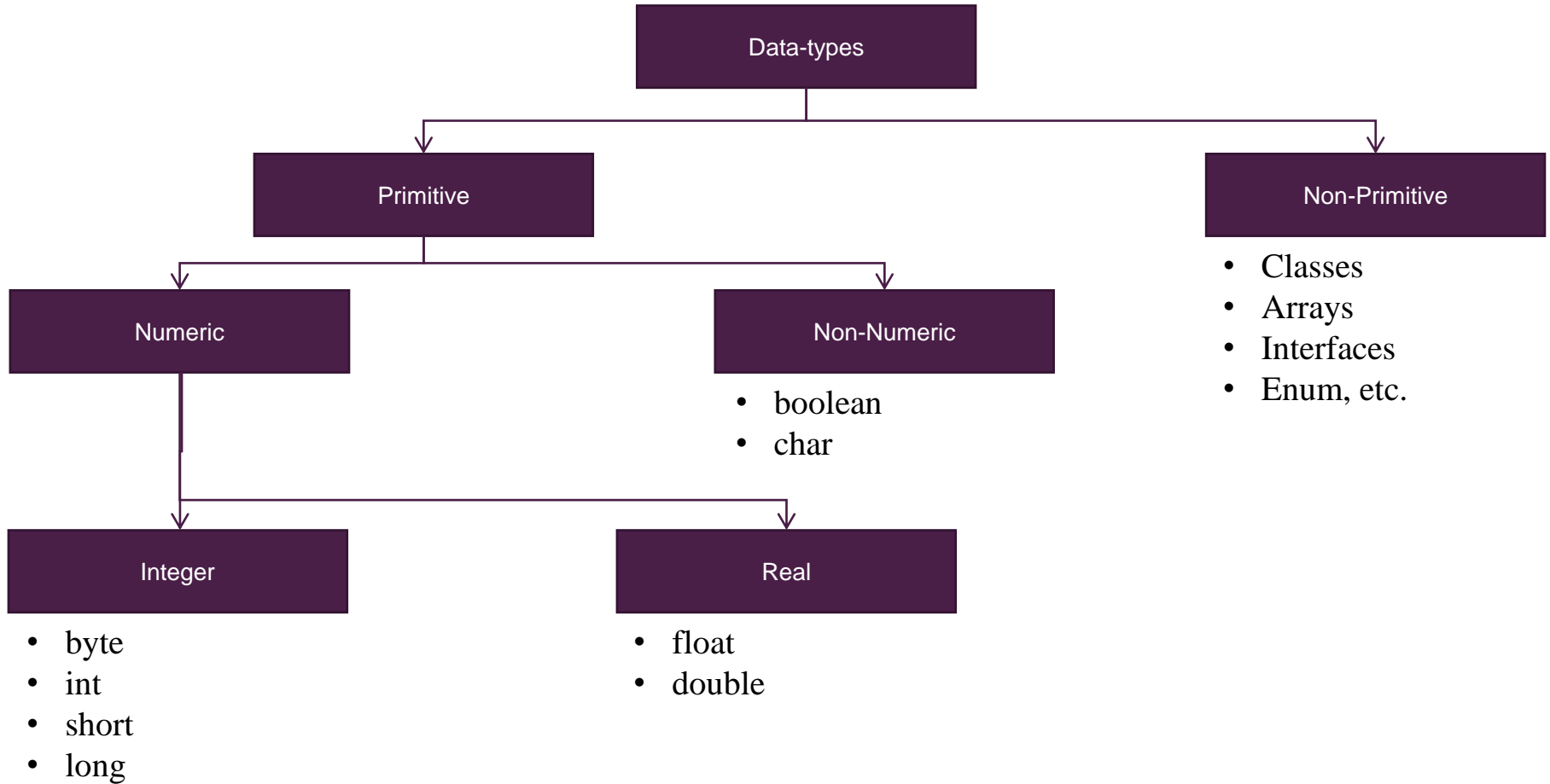
- **instance block:**
  - Instance block is executed when the instance of the class is created.
  - Instance block executes after the execution of static block.
  - 'this' keyword can be used with instance block.
  - Both static and non-static variables can be accessed inside instance block.
  - Instance blocks are basically used for initializing instance variables or calling any instance method.

# instance and static block (contd.)

- **static block:**
  - Static block is executed when the class is loaded.
  - Static block executes before instance block.
  - 'this' keyword cannot be used inside static block.
  - Static blocks are basically used for initializing static variables or calling any static method.

# Data-types

- Data-types are the keyword which defines what type of value a variable may hold.
- There is no 'unsigned' keyword in java.
- There is no 'short int' or 'long int' in java.



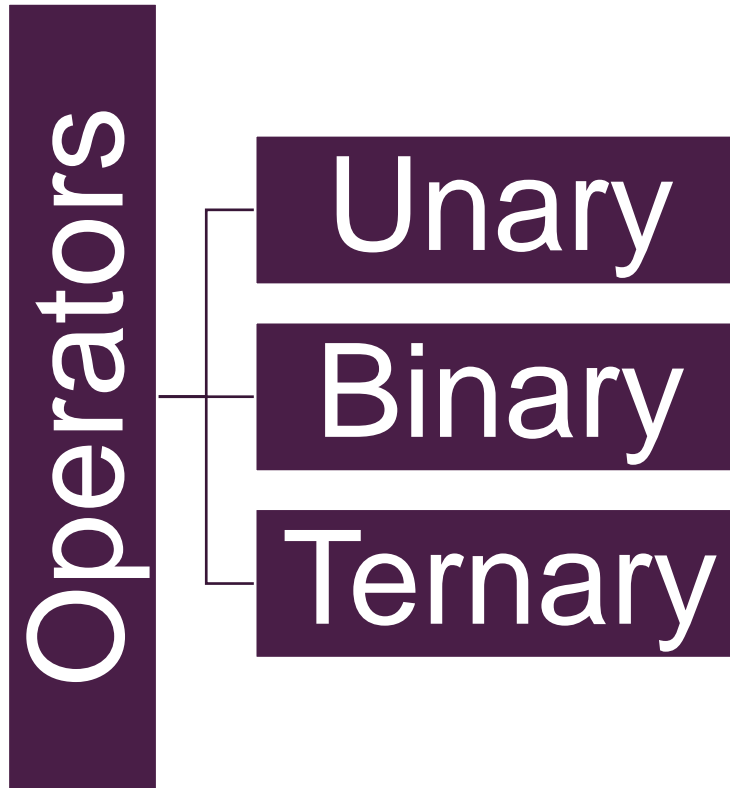
# Literals

A literal is a source code representation of a fixed value. They are represented directly in the code without any computation.

- Literals can be assigned to any primitive type variable.
  - E.g.: `int a = 10;`
- Prefix '0' is used to indicate octal, and prefix '0x' indicates hexadecimal when using these number systems of literals.
  - E.g.: `int decimal=10;`  
`int octal=014;`  
`int hexa=0x10;`



# Types of Operators



# Types of Operators (contd.)

Arithmetic Operators: + - \* / %

Logical Operators: && || !

Relational/Comparison Operators: == != > < >= <=

Bitwise Operators: & | ^ !

Shift Operators: >> << >>>

Conditional/Ternary/Tertiary/Question-mark Operator: int res=(a>b)?a:b;

Assignment/Short-hand Operators: = += -= \*= /= %=

Dot Operator: .

Increment/Decrement Operators:

➤ Pre-increment ++

➤ Pre-decrement --



➤ Post-increment ++

➤ Post-decrement --

# Conditionals statements

- In Java we have four types of conditional statements:
- 
- if statement
- if-else statement
- if-else if statement
- switch statement

# Conditionals statements

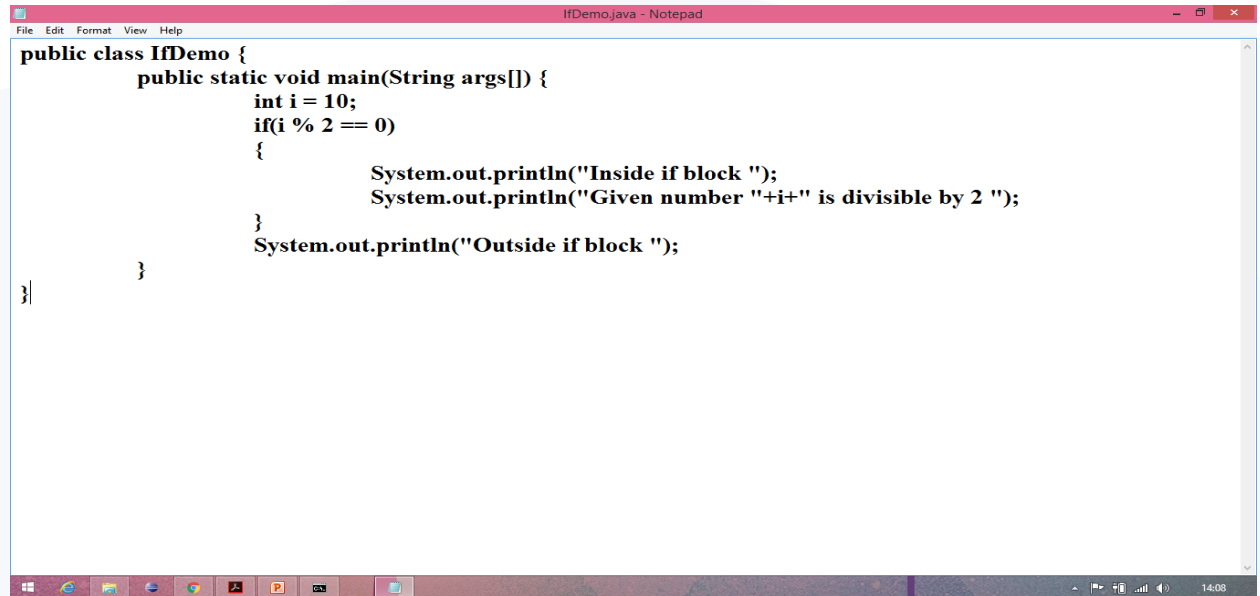
- In Java we have four types of conditional statements:
  - 
  - if statement
  - if-else statement
  - if-else if statement
  - switch statement

# Conditionals statements (contd.)

- **if statement:**
- if statement is used if we want to execute a block of code if the condition is true.
- Syntax:
  - `if(condition)`
  - `{`
  - `block to be executed if the condition is true`
  - `}`

# Conditionals statements (contd.)

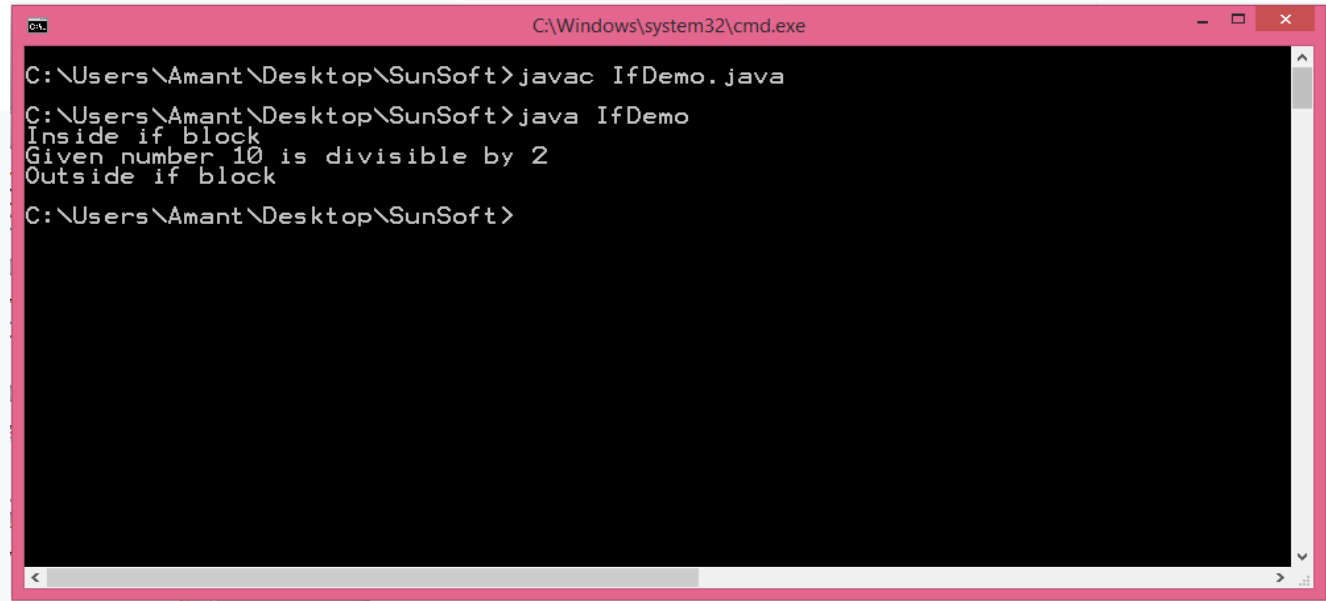
- if statement demo:



```
public class IfDemo {  
    public static void main(String args[]) {  
        int i = 10;  
        if(i % 2 == 0)  
        {  
            System.out.println("Inside if block ");  
            System.out.println("Given number "+i+" is divisible by 2 ");  
        }  
        System.out.println("Outside if block ");  
    }  
}
```

# Conditionals statements (contd.)

- if statement demo:



```
C:\Windows\system32\cmd.exe

C:\Users\Amant\Desktop\SunSoft>javac IfDemo.java
C:\Users\Amant\Desktop\SunSoft>java IfDemo
Inside if block
Given number 10 is divisible by 2
Outside if block
C:\Users\Amant\Desktop\SunSoft>
```

The screenshot shows a Windows Command Prompt window with a pink title bar. The window title is "C:\Windows\system32\cmd.exe". The command prompt shows the following sequence of commands and output:

- Command: `javac IfDemo.java`
- Command: `java IfDemo`
- Output: `Inside if block`
- Output: `Given number 10 is divisible by 2`
- Output: `Outside if block`
- Command: `C:\Users\Amant\Desktop\SunSoft>`

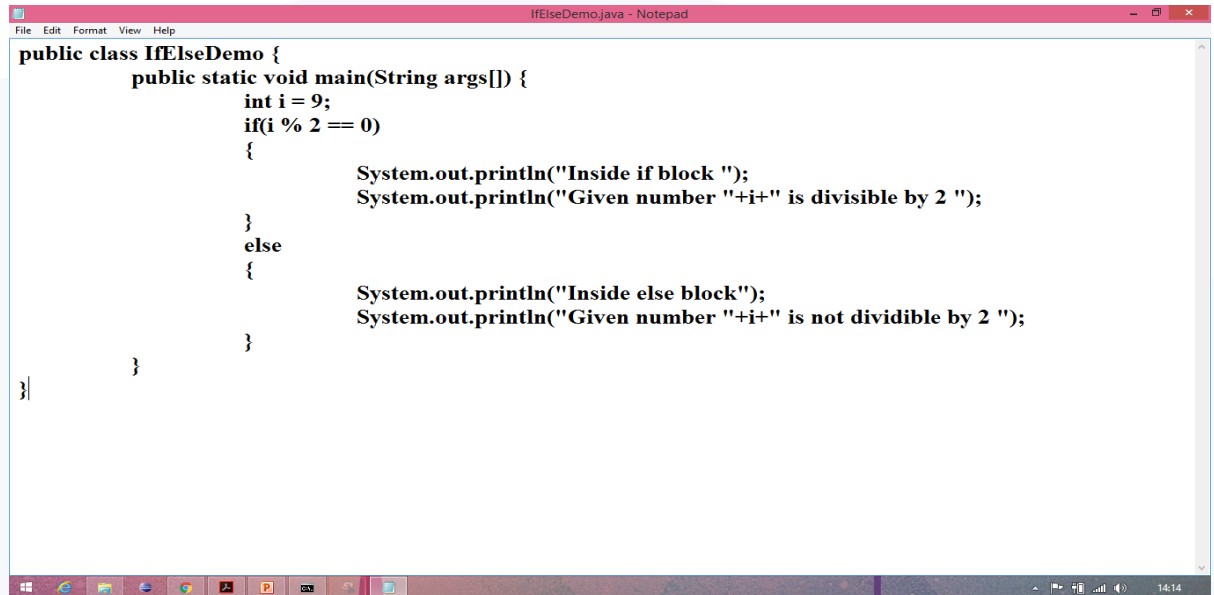
# Conditionals statements (contd.)

- **if else statement:** if else statement is used if we want to execute a block of code if the condition is true otherwise else block is executed.
- Syntax:
- if(condition)
- {
- block to be executed if the condition is true
- }
- else
- {
- block to be executed if the condition is false
- }



# Conditionals statements (contd.)

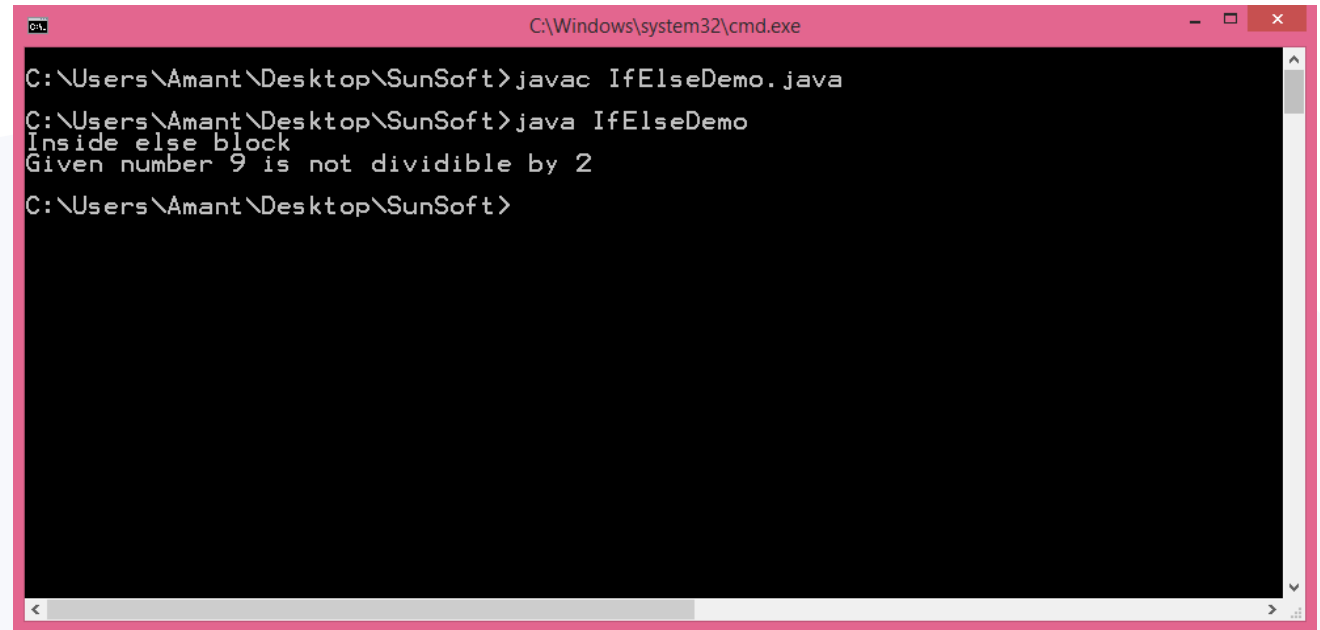
- if else statement demo:



```
public class IfElseDemo {  
    public static void main(String args[]) {  
        int i = 9;  
        if(i % 2 == 0)  
        {  
            System.out.println("Inside if block ");  
            System.out.println("Given number "+i+" is divisible by 2 ");  
        }  
        else  
        {  
            System.out.println("Inside else block");  
            System.out.println("Given number "+i+" is not dividible by 2 ");  
        }  
    }  
}
```

# Conditionals statements (contd.)

- if else statement demo:



```
C:\Windows\system32\cmd.exe

C:\Users\Amant\Desktop\SunSoft>javac IfElseDemo.java
C:\Users\Amant\Desktop\SunSoft>java IfElseDemo
Inside else block
Given number 9 is not dividible by 2
C:\Users\Amant\Desktop\SunSoft>
```

The screenshot shows a Windows command prompt window with a pink title bar. The window title is "C:\Windows\system32\cmd.exe". The command prompt shows the following sequence of commands and output:

- Command: `javac IfElseDemo.java`
- Command: `java IfElseDemo`
- Output: `Inside else block`
- Output: `Given number 9 is not dividible by 2`
- Command: `C:\Users\Amant\Desktop\SunSoft>`

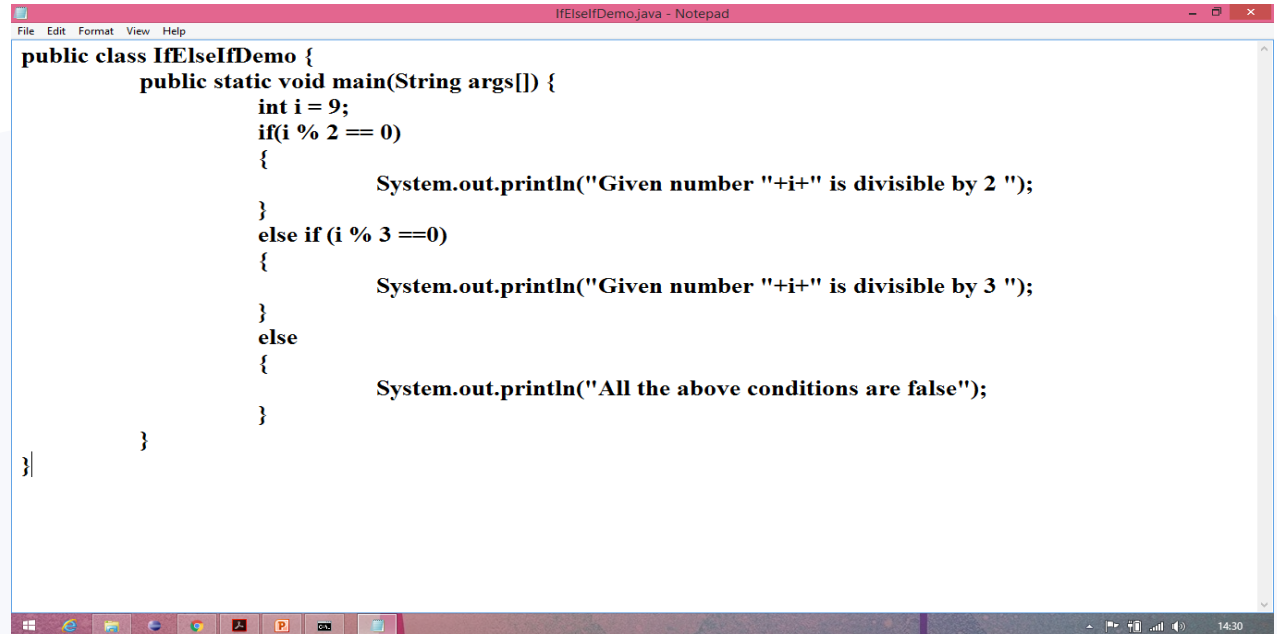
# Conditionals statements (contd.)

- **if-else if statement:** if-else if can be used when we have to check multiple conditions. **Syntax:**

- ```
if(condition)
```
- ```
{
```
- ```
    block to be executed if this condition is true
```
- ```
}
```
- ```
else if(condition)
```
- ```
{
```
- ```
    block to be executed if this condition is true
```
- ```
}
```
- ```
else {
```
- ```
    block to be executed if all the conditions are false
```
- ```
}
```

# Conditionals statements (contd.)

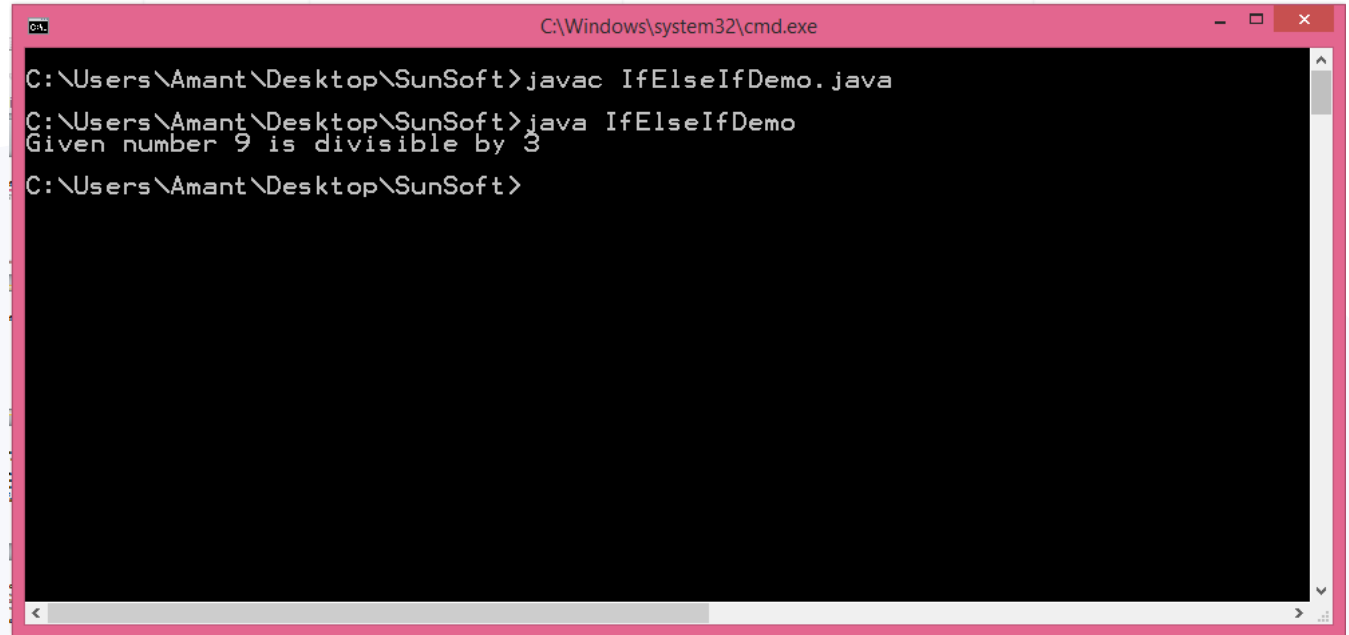
- if-else if statement demo:



```
public class IfElseIfDemo {  
    public static void main(String args[]) {  
        int i = 9;  
        if(i % 2 == 0)  
        {  
            System.out.println("Given number "+i+" is divisible by 2 ");  
        }  
        else if (i % 3 ==0)  
        {  
            System.out.println("Given number "+i+" is divisible by 3 ");  
        }  
        else  
        {  
            System.out.println("All the above conditions are false");  
        }  
    }  
}
```

# Conditionals statements (contd.)

- if-else if statement demo:



```
C:\Windows\system32\cmd.exe
C:\Users\Amant\Desktop\SunSoft>javac IfElseIfDemo.java
C:\Users\Amant\Desktop\SunSoft>java IfElseIfDemo
Given number 9 is divisible by 3
C:\Users\Amant\Desktop\SunSoft>
```

The screenshot shows a Windows command prompt window with a pink title bar. The title bar text is "C:\Windows\system32\cmd.exe". The command prompt shows the following sequence of commands and output:

- Command: `javac IfElseIfDemo.java`
- Command: `java IfElseIfDemo`
- Output: `Given number 9 is divisible by 3`
- Command: `C:\Users\Amant\Desktop\SunSoft>`

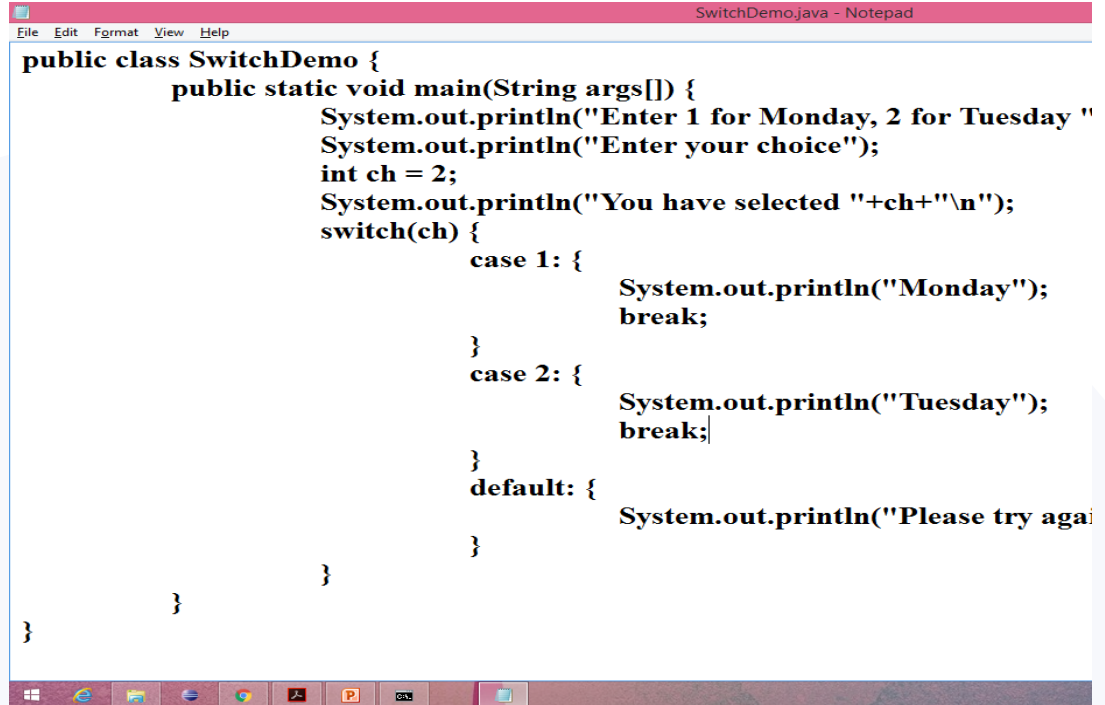
# Conditionals statements (contd.)

- **switch statement:** switch statements can be used when we have to check multiple conditions and execute any one according to user's choice.
- **Syntax:**

```
switch(choice) {  
    •         case 1:{  
                                statements  
    •         }  
    •         case 2:{  
                                statements  
    •         }  
    •         default :{  
                                statements  
    •         }  
}
```

# Conditionals statements (contd.)

- switch statement demo:

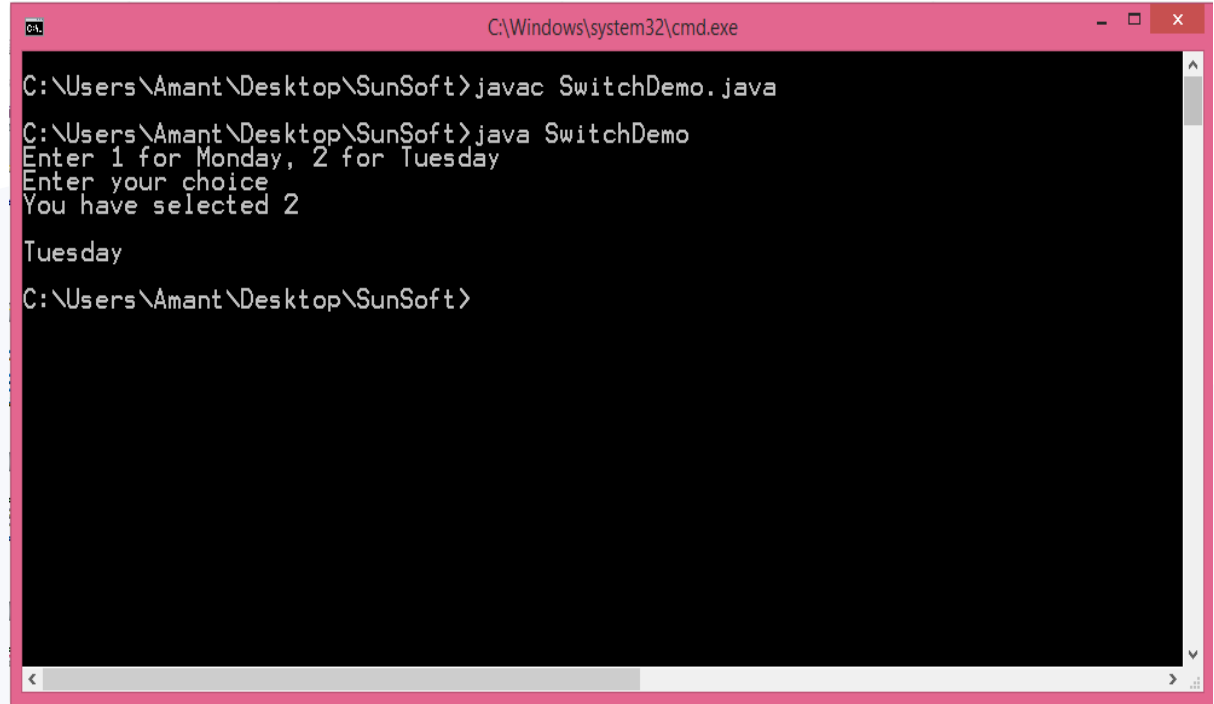
A screenshot of a Notepad window titled "SwitchDemo.java - Notepad". The window contains Java code for a switch statement demo. The code is as follows:

```
public class SwitchDemo {  
    public static void main(String args[]) {  
        System.out.println("Enter 1 for Monday, 2 for Tuesday ");  
        System.out.println("Enter your choice");  
        int ch = 2;  
        System.out.println("You have selected "+ch+"\n");  
        switch(ch) {  
            case 1: {  
                System.out.println("Monday");  
                break;  
            }  
            case 2: {  
                System.out.println("Tuesday");  
                break;  
            }  
            default: {  
                System.out.println("Please try again");  
            }  
        }  
    }  
}
```

The Notepad window has a menu bar with "File", "Edit", "Format", "View", and "Help". The Windows taskbar is visible at the bottom with icons for Start, File Explorer, Edge, Chrome, and other applications.

# Conditionals statements (contd.)

- switch statement demo:



```
C:\Windows\system32\cmd.exe

C:\Users\Amant\Desktop\SunSoft>javac SwitchDemo.java

C:\Users\Amant\Desktop\SunSoft>java SwitchDemo
Enter 1 for Monday, 2 for Tuesday
Enter your choice
You have selected 2

Tuesday

C:\Users\Amant\Desktop\SunSoft>
```



# Java Magic Byte Code

- CA FE: These two bytes are used to indicate that the file is in the Portable Executable (PE) format, which is a common file format used on various computer systems, including Windows.
- BA BE: These two bytes serve as a marker specific to Java. They are used to identify the file as a Java class file within the PE format. When the JVM encounters these bytes, it recognizes the file as a valid Java class file and proceeds to load and execute it.

# Java Architecture

- Java Source Code
- Java Compiler
- Bytecode
- Java Virtual Machine (JVM)
- Java Class Loader
- Execution Engine
- Java Native Interface (JNI)
- Java Standard Library (Java API)
- Java Development Kit (JDK)
- Java Runtime Environment (JRE)
- Application or Applet



Any or all third-party material that are available in this content are provided "as is" without warranty of any kind, either expressed or implied and such material is to be used at your own risk. Each third-party content provider is solely responsible for any content it provides, including any warranties (to the extent that such warranties have not been disclaimed), for any claims you may have relating to that content or your use of that content."

[xebia.com](https://xebia.com)

