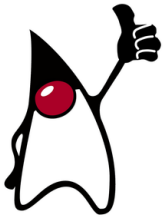
LavaTech Technology

Core Java



Call: 096073 31234
Email: info@lavatechtechnology.com
Website: https://lavatechtechnology.com
Address: Pune, Maharashtra

PUBLISHED BY LAVATECH TECHNOLOGY

lavatechtechnology.com

January 2022

YOU CAN

TOTALLY

DO THIS!

Contents

LINUX PADHO!

JOB KI CHINTA MAAT KARO

1. Introduction to Java

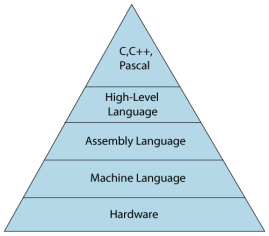
1.1 Getting started with Java

In this section, you are going to learn:

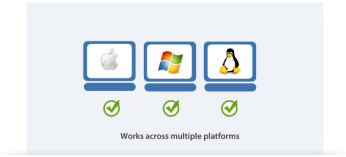
1. **What is Java?**
2. **Uses of Java**
3. **Problem with C and C++**
4. **Advantages of Java over C++**
5. **How does program execution works?**
6. **The way Java works**
7. **Compiler V/S JVM**
8. **JVM components**
9. **History of Java**

1.1.1 What is Java?

- Java is a popular **high-level programming language** that is **platform-independent**, **object-oriented** and **open source**.
- Let's understand each bold words in detail:
  - **High level programming langauge**: Language that are in english and can be understood by humans.



  - **Platform-independent**: Java code can run on any platform (i.e OS) using Java Virtual Machine (JVM).



  - **Object-Oriented**: Java is built around objects.

  - **Open source**: Java source code is available for anyone to view and modify.

1.1.2 Uses of Java

- **Enterprise-level applications** - Eg: SAP, IBM websphere, Salesforce, Oracle E-Business suite

- **Web applications** - Eg: LinkedIn, Netflix, Twitter, Amazon, Airbnb

- **Mobile applications** - Eg: Instagram, WhatsApp, Google Maps, Uber

- **Games** - Eg: Minecraft, RuneScape, Puzzle Pirates

- **Financial Applications** - Eg: Quicken, Bloomberg Terminal

- **Scientific Applications** - Eg: BioJava, ImageJ

1.1.3 Problem with C and C++

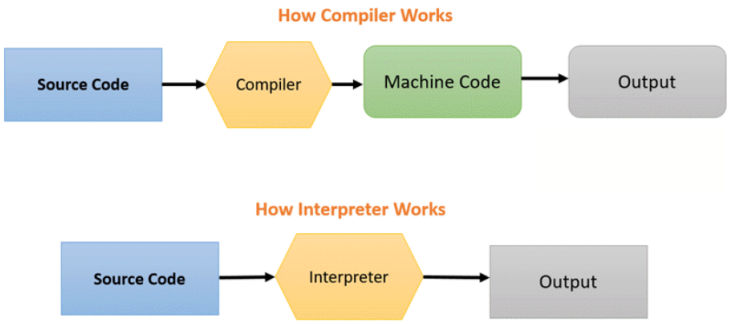
- **Original idea for Java was not the Internet!**
- Java was created to be **platform-independent language** that can be embedded in various consumer electronic devices, eg: **microwave ovens and remote controls**.
- The trouble with C and C++ is that they are **compiled for a specific target**.
- A full C++ compiler targeted for a specific CPU is needed to compile a C++ program for different CPU.
- The problem is that **compilers are expensive and time-consuming to create**.
- James Gosling (founder of Java) began work on a portable, platform-independent language that could be used to produce code that would run on any CPUs.
- This led to the creation of Java.

1.1.4 Advantages of Java over C++

- **Security**: No danger of reading bogus data when accidentally going over the size of an array.
- **Automatic memory management**:
  - Garbage collector allocate and deallocate memory for objects.
  - Pointers are not necessary.
- **Simplicity**:
  - No pointers, unions, templates, structures, multiple inheritance.
  - Anything in Java can be declared as a class.
- **Support for multithreaded execution**: Support development of multithreaded software.
- **Portability**: Support WORA (Write it once, run it anywhere), using Java virtual machine (JVM).

1.1.5 How does program execution works?

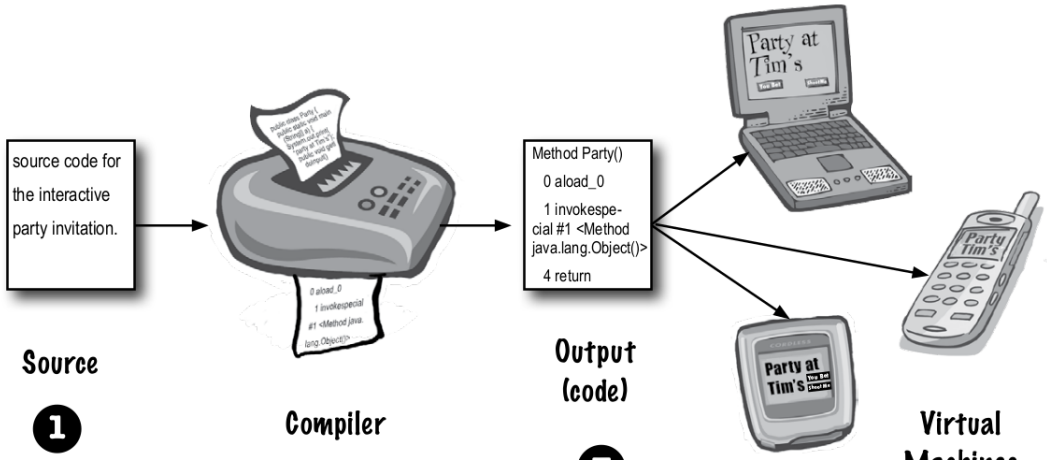
- **Compiler:**
  - A compiler is a **software program that converts source code written in a high-level programming language into machine code**, which can be executed by a computer.
  - The compiler performs:
    - Syntax analysis
    - Semantic analysis
    - Bytecode generation
- **Interpreter:**
  - An interpreter is a program that reads and executes code **line-by-line**, without the need to compile the entire program beforehand.
  - Interpreters run code in a virtual environment, where each line of code is executed as soon as it is read.



- **Java Virtual Machine (JVM):**
  - The JVM is a virtual machine that is responsible for executing Java bytecode.
  - The JVM is an example of an interpreter, as it reads Java bytecode and executes it line-by-line.
  - The JVM performs:
    - Memory management
    - Security
    - Garbage collection

1.1.6 The way Java works

So, what exactly happens to developer-written Java code until the actual execution?



Source

1

Create a source document. Use an established protocol (in this case, the Java language).

Compiler

2

Run your document through a source code compiler. The compiler checks for errors and won't let you compile until it's satisfied that everything will run correctly.

Output (code)

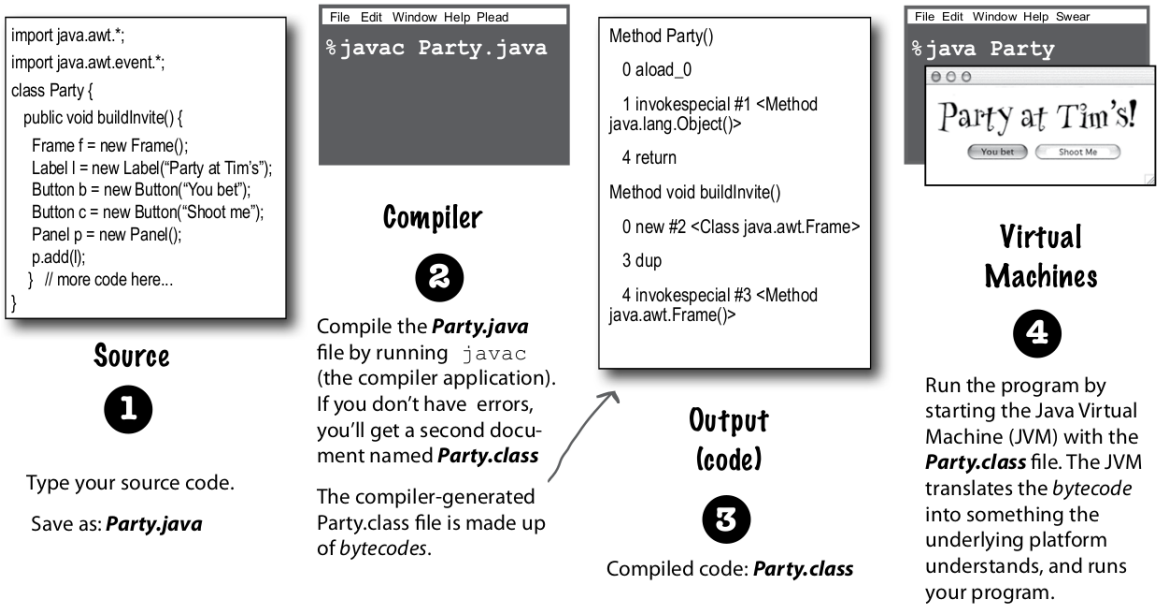
3

The compiler creates a new document, coded into Java ***bytecode***. Any device capable of running Java will be able to interpret/translate this file into something it can run. The compiled bytecode is platform-independent.

Virtual Machines

4

Your friends don't have a physical Java Machine, but they all have a ***virtual*** Java machine (implemented in software) running inside their electronic gadgets. The virtual machine reads and *runs* the bytecode.



Source

1

Type your source code.

Save as: ***Party.java***

Compiler

2

Compile the ***Party.java*** file by running `javac` (the compiler application). If you don't have errors, you'll get a second document named ***Party.class***

The compiler-generated Party.class file is made up of *bytecodes*.

Output (code)

3

Compiled code: ***Party.class***

Virtual Machines

4

Run the program by starting the Java Virtual Machine (JVM) with the ***Party.class*** file. The JVM translates the *bytecode* into something the underlying platform understands, and runs your program.

1.1.7 Compiler V/S JVM

Compiler and JVM battle over the question, "Who's more important?"

Ans: Go through below discussion to find the answer:

JVM: I am Java. I'm the guy who actually makes a program run. The compiler just gives you a file in bytecode after checking it's syntax. I'm the one who run it.

Compiler: Excuse me? Without me, you would have to translate everything from source code and be very very slow!

JVM: Your work is not important. A programmer could just write bytecode by hand. You might be out of a job soon, buddy.

Compiler: That's arrogant. A programmer writing bytecode by hand is next to possible, some scholars might write, not everyone!

JVM: But you still didn't answer my question, what you actually do?

Compiler: Remember that Java is a strongly-typed language, I can't allow variables to hold data of the wrong type. This is a security feature, implement by ME!

JVM: Your type checking is not very strict! Sometimes people put the wrong type of data in an array of different type.

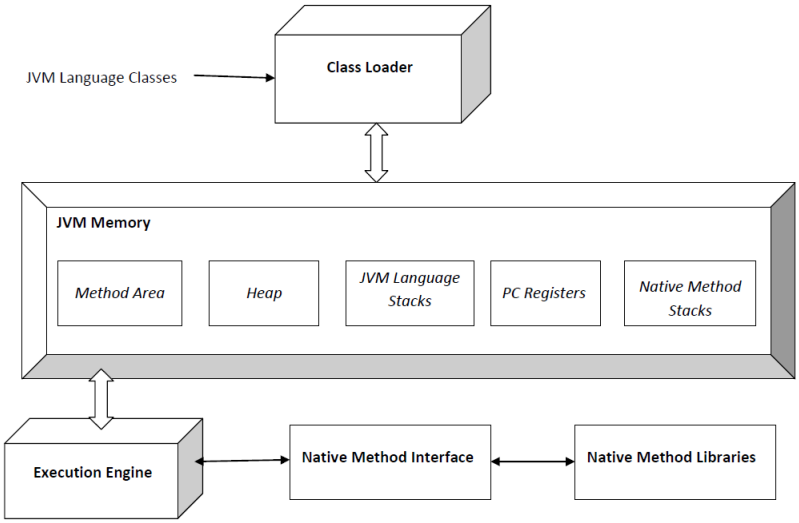
Compiler: Yes, that can emerge at runtime that only you can catch to allow dynamic binding. But my job is to stop anything that would never succeed at runtime.

JVM: OK. Sure. But what about security? Look at all the security stuff I do! You just perform silly syntax checking.

Compiler: Listen, I'm the first line of defense. I also prevents access violations, such as code trying to invoke a private method. I stop people from touching code they're not meant to see.

JVM: Whatever. I have to do that same stuff too!

1.1.8 JVM components



- **Class Loader:** Responsible for loading the class files into the memory of the JVM.
- **Execution Engine:** Responsible for executing the bytecode that is loaded into the memory. It includes:
  - **Interpreter:** Reads and executes the bytecode one instruction at a time.
  - **JIT compiler:** Compiles the bytecode into machine code for fast execution.
- **Garbage Collector:** It periodically frees up the memory that is not used by the Java application.
- **Runtime Data Area:** It is memory space allocated by the JVM for the execution of the Java application. It includes:
  - Method area
  - Heap
  - Stack
  - PC registers
- **Native Method Interface (JNI):** Allows Java code to call code written in other programming languages like C and C++. The JNI allows Java applications to interact with OS and hardware.

1.1.9 History of Java

What year Java was invented?

Ans: 1995

What company invented Java?

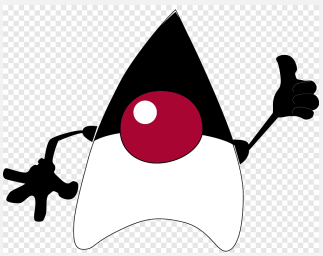
Ans: Sun Microsystems

Who is founder of Java?

Ans: James Gosling

What is Java mascot?

Ans: A cartoon character named Duke



What is the original name of Java ?

Ans: "Oak" after the oak tree that was outside Gosling's office.

What was the reason for changing original name?

Ans: "Oak" was already trademarked

What is the inspiration behind Java's name?

Ans: Java language is named after coffee grown on the Indonesian island

What is original Java logo?

Ans: Original logo:

Who has the current ownership of Java?

Ans: Oracle acquired Java in 2009

What are the Java versions?

Ans: Below are the Java version details:

Version	Year
JDK Alpha and Beta	1995
JDK 1.0	Jan, 1996
JDK 1.1	Feb, 1997
J2SE 1.2 or **Java2** (codename: Playground)	Dec, 1998
J2SE 1.3 or **Java2** (codename: Kestrel)	May, 2000
J2SE 1.4 or **Java2** (codename: Merlin)	Feb, 2002
J2SE 1. 5 or **Java5** (codename: Tiger)	Sep, 2004
Java SE 1.6 or **Java6** (codename: Mustang)	Dec, 2006
Java SE 1.7 or **Java7** (codename: Dolphin)	July, 2011
Java SE 1.8 or **Java8** (codename: Spider)	(18th March, 2014)
Java SE 1.9 or **Java9**	September, 2017
Java 10	March, 2018
Java SE 11	September 2018
Java SE 12	March 2019
Java SE 13	September 2019
Java SE 14	March 2020
Java SE 15	September 2020
Java SE 16	March 2021
Java SE 17	September 2021

Why is Java 2 consider very significant in history of Java?

Ans: Starting **Java 2**, it is composed of three parts:

- **J2SE (Java 2 Platform, Standard Edition) or JSE**, a computing platform for the development and deployment of portable code for **desktop and server environments**.
- **J2EE (Java 2 Platform, Enterprise Edition) or JEE**, extending Java SE with enterprise features such as **distributed computing and web services**.
- **J2ME (Java 2 Platform, Micro Edition) or JME**, a computing platform for **embedded and mobile devices**.

Other major highlights of this release:

- **JIT compiler** became part of JVM (means turning code into executable code became a faster operation).
- **Swing graphical API** was introduced as alternative to AWT.
- Java collections framework (for working with sets of data) was introduced.

I see Java 2 and Java 5.0, but was there a Java 3 and 4? And why is it Java 5.0 but not Java 2.0?

Ans: The joys of marketing...

- When the version of Java shifted from 1.1 to 1.2, the changes to Java were so many that the marketers decided a whole new "name", so they started calling it Java 2, even though the actual version of Java was 1.2.
- But versions 1.3 and 1.4 were still considered Java 2.
- There never was a Java 3 or 4.
- Beginning with Java version 1.5, the marketers decided a new name was needed.
- The next number in the name sequence would be "3", but calling Java 1.5 Java 3 seemed more confusing, so they decided to name it Java 5.0 to match the "5" in version "1.5".

1.2 Installing Java and IDE

In this section, you are going to learn:

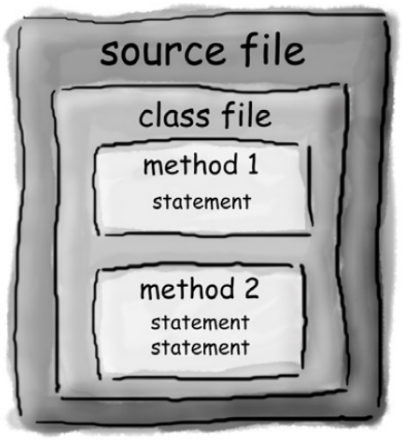
1. **Java installation**
2. **IDE installation**
3. **Our first Java program**
4. **Using JShell**

1.2.1 Java installation

1.2.2 IDE installation

1.2.3 Our first Java program

- Code structure in Java:



Put a class in a source file.

Put methods in a class.

Put statements in a method.

What goes in a source file?

Ans:

- A source code file with the **".java"** extension holds one class definition.
- The source file name should be **"classname.java"**.
- The class represents a piece of your program.
- The class must go within a pair of curly braces.
- Eg: Below code name will be Dog.java and class name will be Dog -

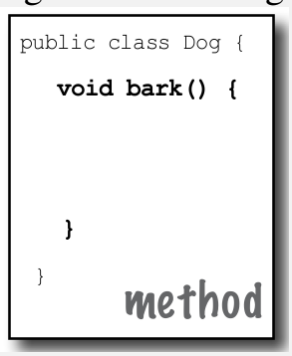
```
public class Dog {

}
```

class

What goes in a class?

Ans:

- A class has one or more methods.
- Your methods must be declared inside a class.
- Eg: The class Dog contains a method called bark()



What goes in a method?

Ans:

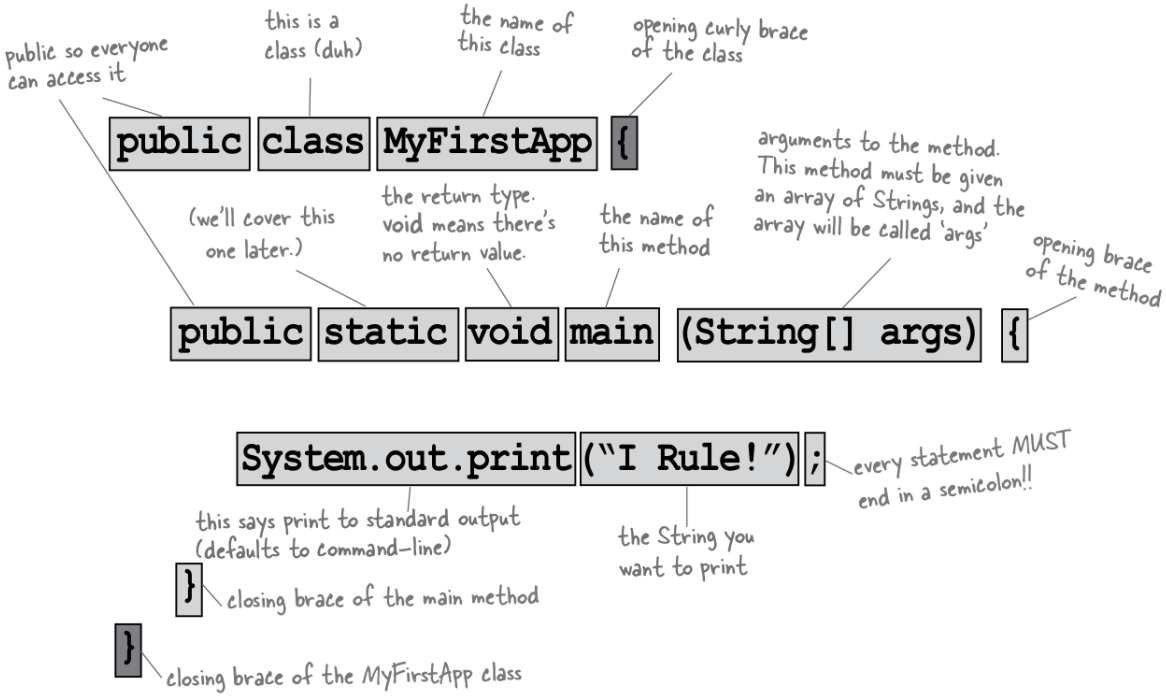
- Method code is basically a set of statements.
- Method kind of like a function or procedure.
- When the JVM starts running, it looks for the method inside the class that looks exactly like:

Code:

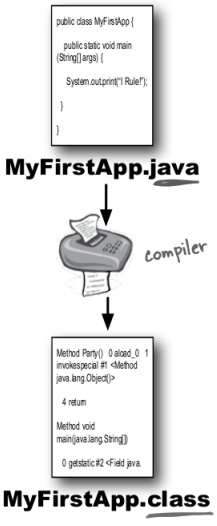
```
public static void main (String[] args) {
    // your code goes here
}
```

- JVM runs everything inside { } of your main method.
- Every Java application has to have at least one class, and at least one main method (not one main per class; just one main per application).

Overall, a basic Java program would look something like below:



Running your Java Code:




```
public class MyFirstApp {

   public static void main (String[] args) {
     System.out.println("I Rule!");
     System.out.println("The World");
   }

}
```

1 Save

`MyFirstApp.java`

2 Compile

```
javac MyFirstApp.java
```

3 Run

```
File Edit Window Help Scream
%java MyFirstApp
I Rule!
The World
```

1.2.4 Executing our first Java program

- Using simple text editor:

MyFirstApp.java

```
public class MyFirstApp {
    public static void main(String[] args) {
        System.out.println("I Rule!");
        System.out.println("The World");
    }
}
```

Command:

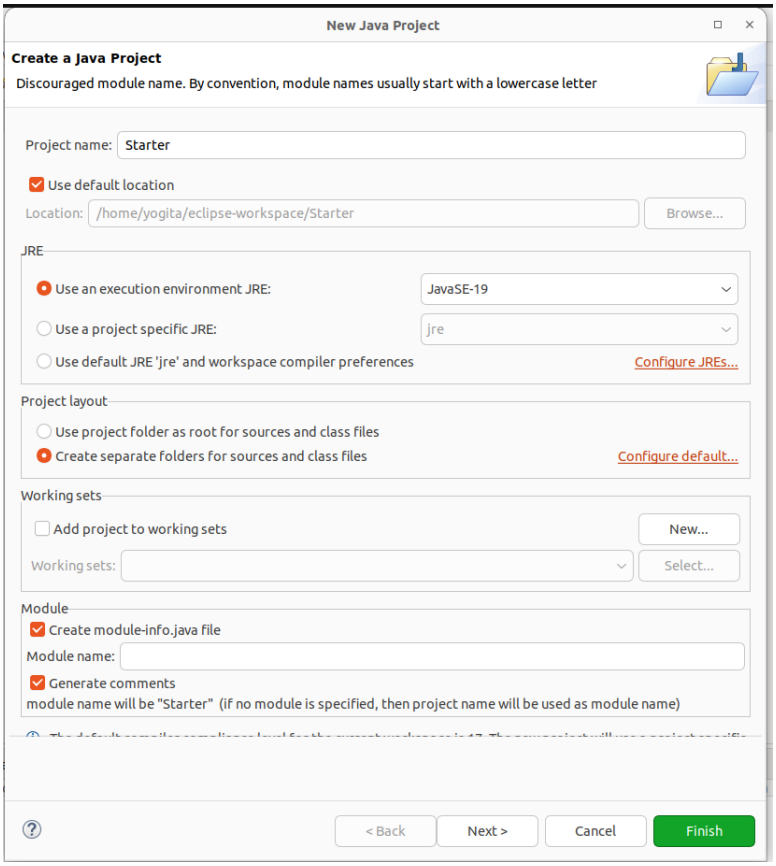
```
javac MyFirstApp.java
java MyFirstApp
```

Output:

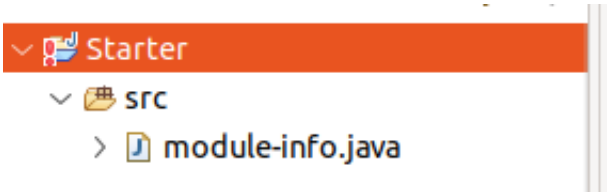
```
I Rule!
The World
```

- Using Eclipse:
  1. Create New Project:

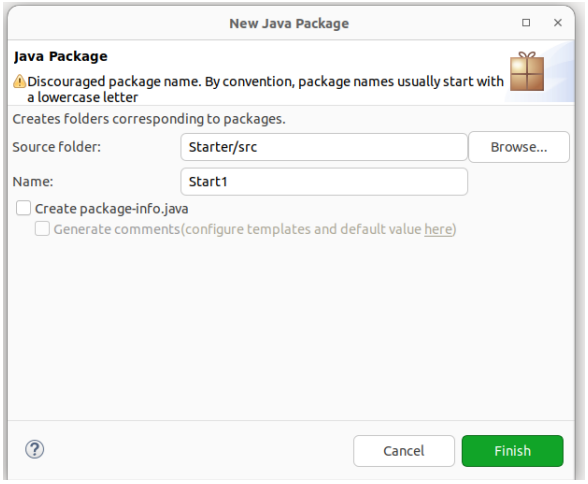
     File -> New -> Java Project -> Add "Starter" as project name



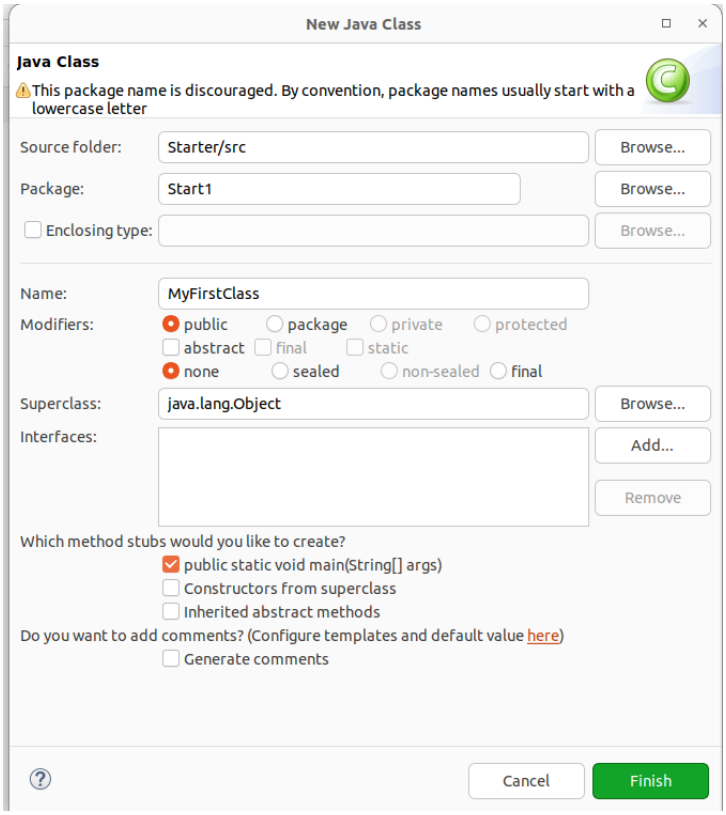
  2. This will create directory structure as shown below:



3. Right click the src -> Select “Package” -> Add “Start1” as package name as shown below:



4. Right click the “Start1” -> Select New -> Class -> Add “MyFirstClass” as classname as shown below:

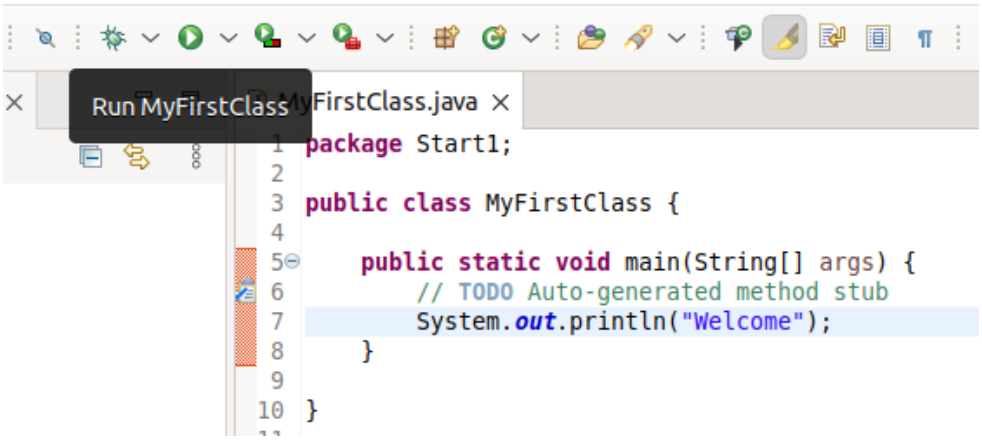


5. This will create a class with below structure:

Code:

```
package Start1;
public class MyFirstClass {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.println("Welcome Back!");
    }
}
```

6. Execute the code by pressing the run button:



1.2.5 Using JShell

- JShell is a Read-Eval-Print Loop (REPL)
- It evaluates declarations, statements, and expressions as they are entered, and then it immediately shows the results.
- It was introduced in Java 9.
- To use jshell, type "jshell" command in command prompt:

Command:

```
$ jshell
| Welcome to JShell – Version 11.0.18
| For an introduction type: /help intro

jshell> int i=42;
i ==> 42

jshell> float j=3.4f;
j ==> 3.4

jshell> i+j
$3 ==> 45.4

jshell> String text = "Welcome To World of Java";
text ==> "Welcome To World of Java"

jshell> text.toUpperCase()
$5 ==> "WELCOME TO WORLD OF JAVA"
```

- To display all variables declared:

Command:

```
jshell> /vars
| int i = 42
| float j = 3.4
| float $3 = 45.4
| String text = "Welcome To World of Java"
| String $5 = "WELCOME TO WORLD OF JAVA"
```

- To save all valid statements of Jshell to a file:

Command:

```
jshell> /save filename.java


jshell> /exit
| Goodbye
```

- Content of filename.java:

filename.java

```
int i=42;
float j=3.4f;
i+j
String text = "Welcome To World of Java";
text.toUpperCase()
```

- You can open the file back in the jshell using open command:

Command:

```
jshell> /open filename.java


jshell> i
i ==> 42
```

Small steps every day.....

2. Java Language Fundamentals

2.1 Identifiers, variables and more

In this section, you are going to learn:

1. **Java identifiers**
2. **Java grammar**
3. **Java variable**
4. **Literals**
5. **Java reserved words or keywords**
6. **Java comments**
7. **How objects can change your life?**

2.1.1 Java identifiers

A Java identifier is a name of a variable, function, class, module or other object.
Eg:

```
package Starter;

public class Test {
    public static void main(String[] args) {
        int x=999;
        System.out.println(x);
    }
}
```

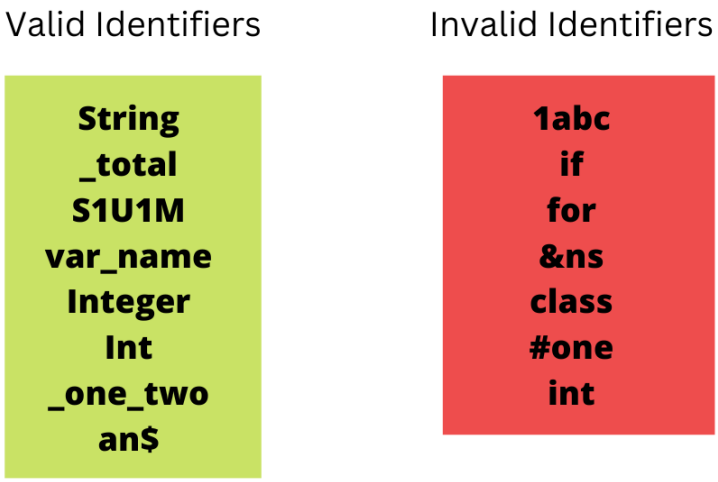
In above code, there are total 6 identifiers:

- Starter - name of package
- Test - name of class
- main - name of function
- String - name of class
- args - name of object
- x - name of integer variable

Rules of identifiers:

- Names can contain A-Z, a-z, 0-9, _, and $ signs.
- Names cannot begin with number.
- Names are case sensitive ("myVar" and "myvar" are different variables).
- Reserved words (like Java keywords, such as int or boolean) cannot be used as names.
- Names can be of any length, but it's not recommended to have big names.

- Developers should declare identifiers using the **Camel case** writing style (e.g., StringBuilder, isAdult)



2.1.2 Java grammar

- Java is **case sensitive**.
- **Block delimiters**: Except for import, package, interface (or @interface), enum and class declarations, everything else in a Java source file must be declared between curly brackets ().
- Code lines are ended in Java by the **semicolon (;) symbol**

2.1.3 Java variable

- The variable is the basic unit of storage in a Java program.

Syntax:

```
type identifier = value;
or
type identifier = value, identifier = value, identifier = value ... ;
```

Here,

- type = datatype or name of class or interface
- identifier = name of variable

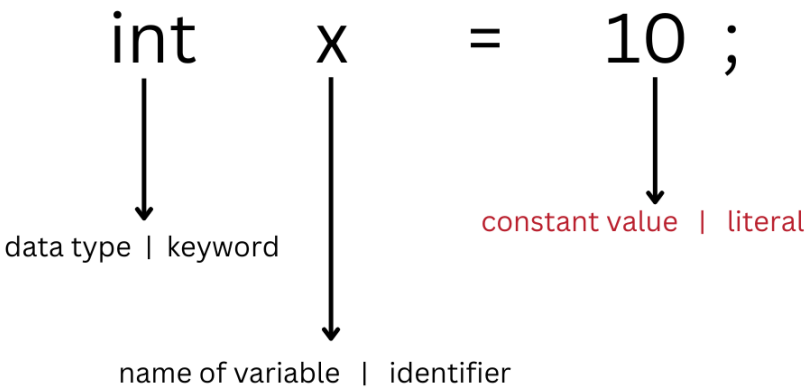
To declare more than one variable of the specified type, use a comma-separated list.

New.java

```
class New {
    public static void main(String[] args) {
        int a, b, c;
        int d = 3, e, f = 5;
        byte z = 22;
        double pi = 3.14159;
        char x = 'x';
    }
}
```

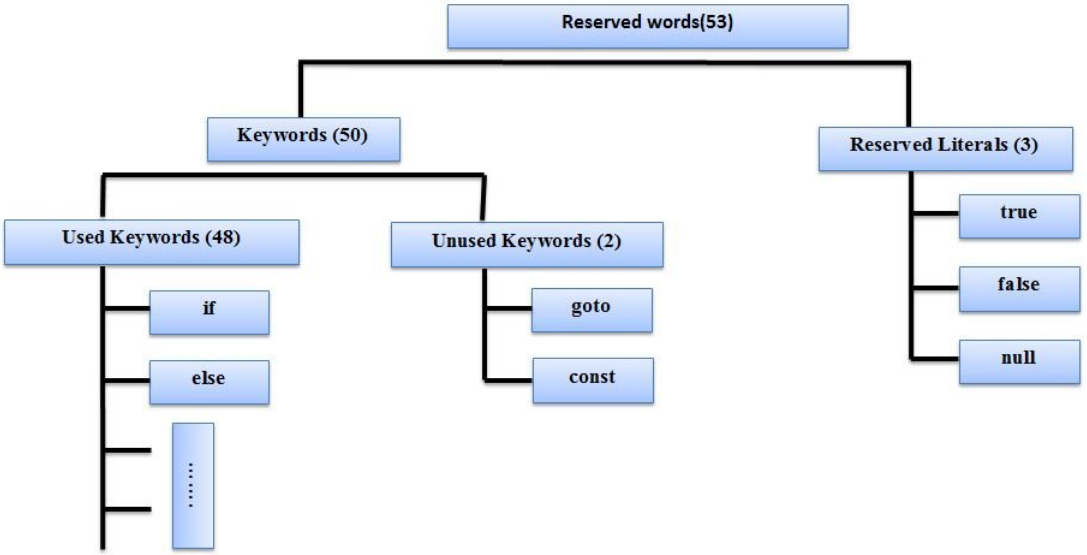
2.1.4 Literals

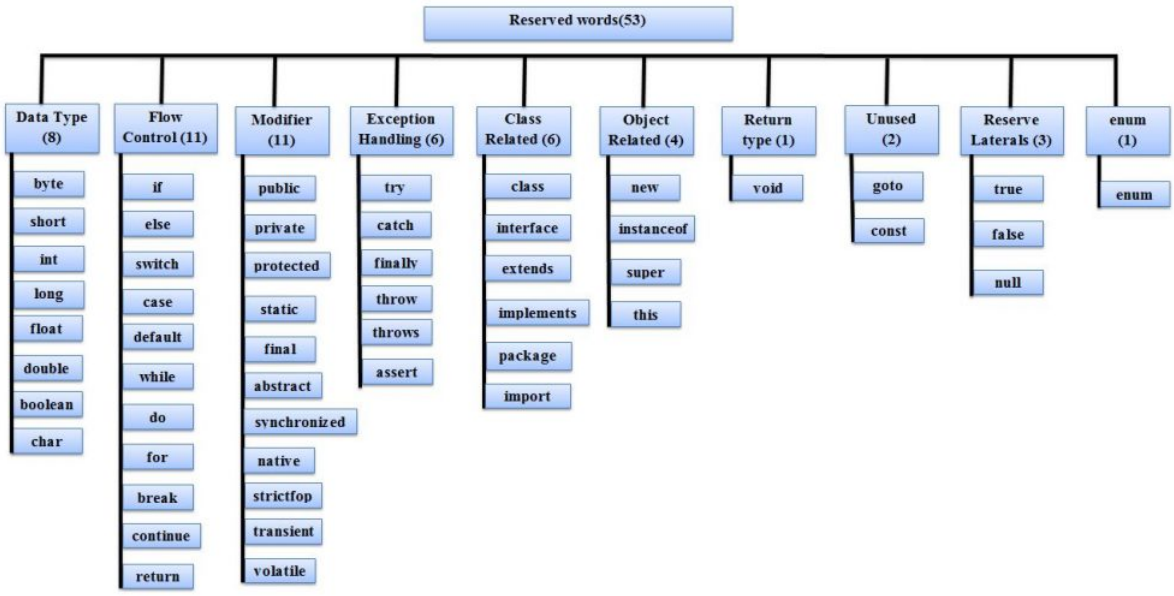
- A constant value which can be assigned to the variable is called **literal**
- Eg:



2.1.5 Java reserved words or keywords

- Words having **predefined meaning**
- Java 17 has a total of 60 reserved words.





Note:

const and **goto** are not used anymore!

2.1.6 Java Comments

- Java comments refer to text that are not considered part of the code execution and ignored by the compiler.
- 3 ways to add comments:
  - **//** : Used for single line comments:

Code:

```
// testing
```

  - **/** ... ***/**: Javadoc comments, special comments that are exported using special tools into the documentation of a project called Javadoc API

Code:

```
/**
* Returns the sum of two integers.
* @param a the first integer to add
* @param b the second integer to add
* @return the sum of a and b
*/
public int add(int a, int b) {
    return a + b;
}
```

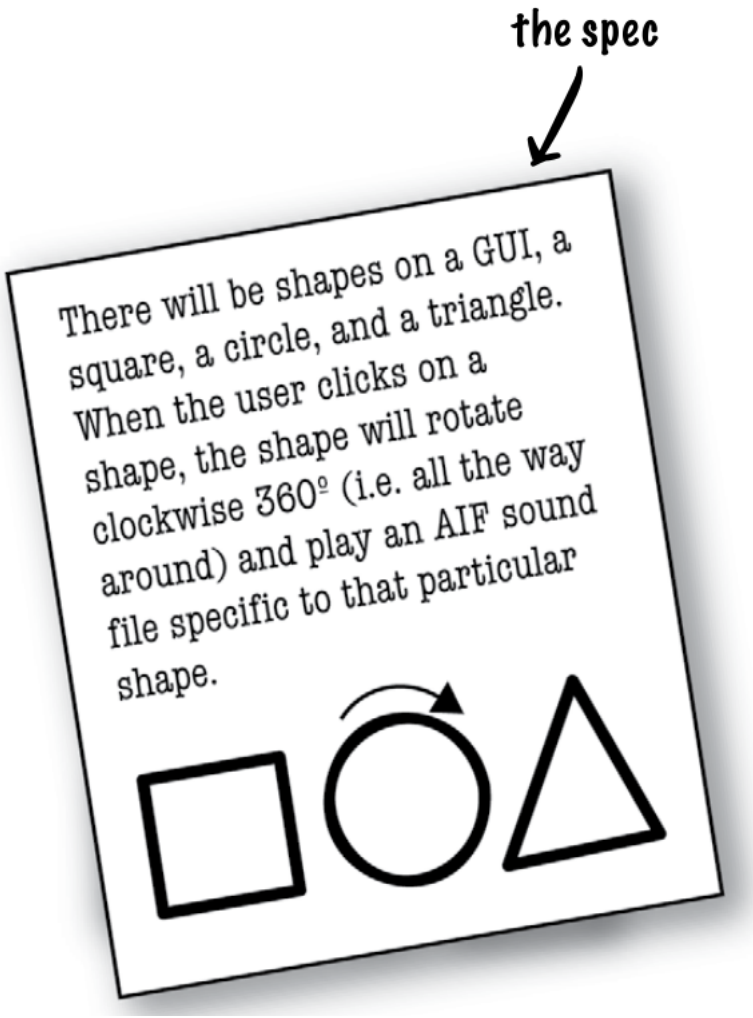
  - **/* ... */** : used for multiline comments

Code:

```
/* This is
a multi line
statement */
```

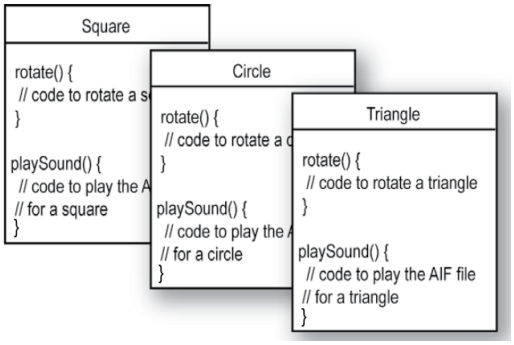
2.1.7 How Objects Can Change Your Life?

- So far, we put all of our code in the **main() method**.
- **That's not exactly object-oriented**.
- Leave the procedural world behind, get the out of main(), and start making some objects of our own.
- We'll look at what makes object-oriented (OO) development in Java so much fun.
- Let's understand this with a use-case:
  Creat a Shape Application with below requirement:

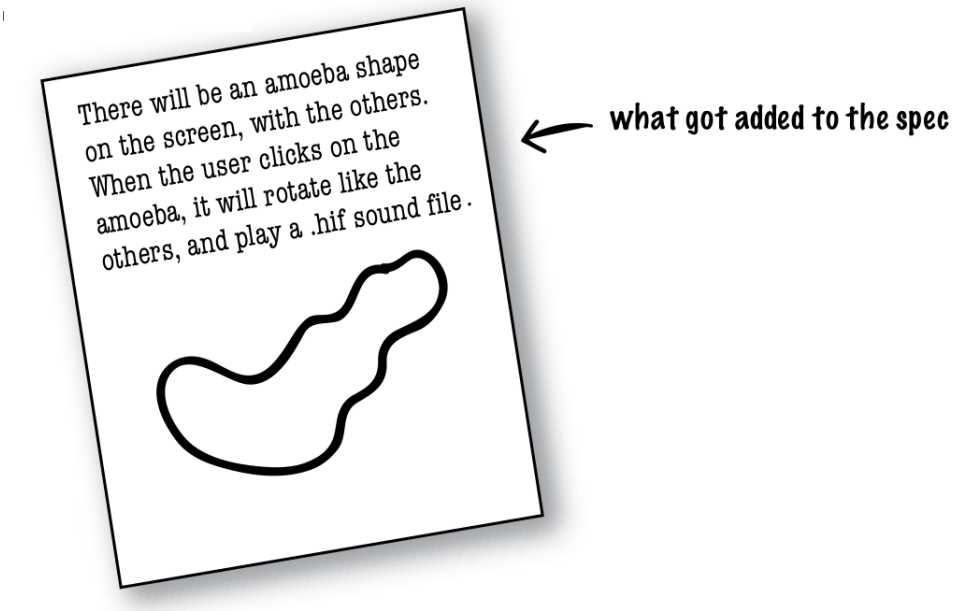


- For this, Raj followed Procedure-Oriented approach while Ram followed Object-Oriented approach to develop the code.

Raj's Procedure-Oriented approach	Ram's Object-Oriented approach

Code:
rotate(shapeNum) {
//code here
}
playSound(shapeNum) {
//code here
}
	

- But wait! There's been a spec change.



- In order to reflect the new requirements, here's what Raj and Ram decided to do:

Back in Raj's cube	At Ram's laptop in a cafe

Raj will need to make changes in existing code and perform the testing again:
 	
Code:
`rotate(shapeNum) { //new changes to add amoeba }`
`playsound(shapeNum) { //add change for amoeba }`
	Ram just need to create one more class called "Amoeba":
	
	Amoeba
	`rotate() { // code to rotate an amoeba }`
	`playSound() { // code to play the new // .hif file for an amoeba }`

So what do you like about OO?

Ans:

- Design software as per real world usage.
- New changes can be incorported easily.
- Not messing around with code already tested, just to add a new feature.
- Data & methods that operate on that data are together in one class.
- Code can be re-used in other applications.

Let's dig a bit deeper into OOP's

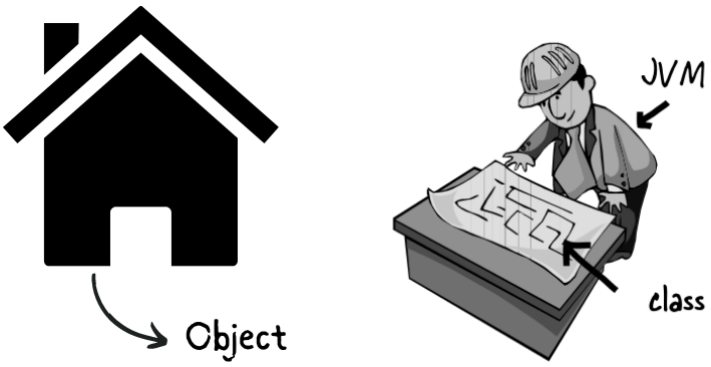
2.2 Introductions to OOPs

Object-**O**riented **P**rogramming (OOP) is a programming paradigm that revolves around the concept of "objects".

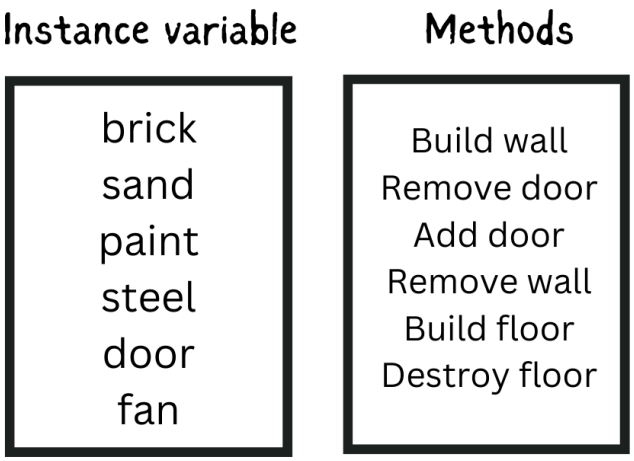
In Java, OOP is implemented through the use of classes and objects.

Class & Object

- A class is a blueprint for an **object**.
- It tells the JVM how to make an object of that particular type.
- An Object is real world entity. It is an instance of class.



- A class consists of instance **variable and methods**:
  - **Instance variable:** Represents the object data.
  - **Methods:** Things an object can do are called methods.

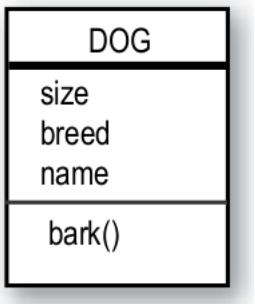


- Java code for class and object would look something like below:

```
class Dog {

  int size;
  String breed;
  String name;

  void bark() {
    System.out.println("Ruff! Ruff!");
  }
}
```

instance variables

a method




```
class DogTestDrive {
   public static void main (String[] args) {
     Dog d = new Dog();
     d.size = 40;
     d.bark();
   }
}
```

make a Dog object

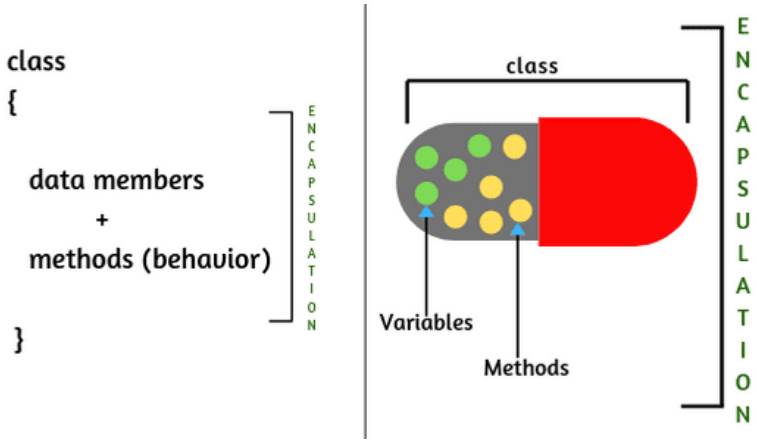
use the dot operator (.) to set the size of the Dog

and to call its bark() method

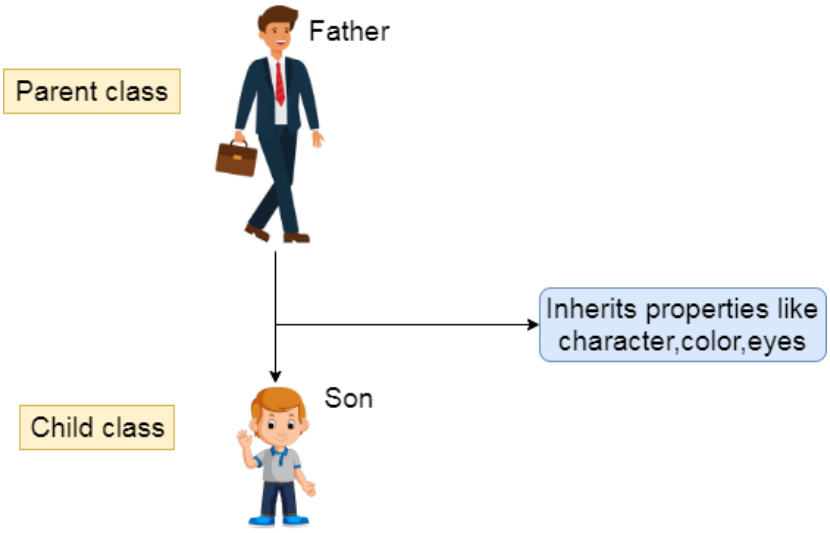
dot operator

2.2.1 Pillars of OOPs

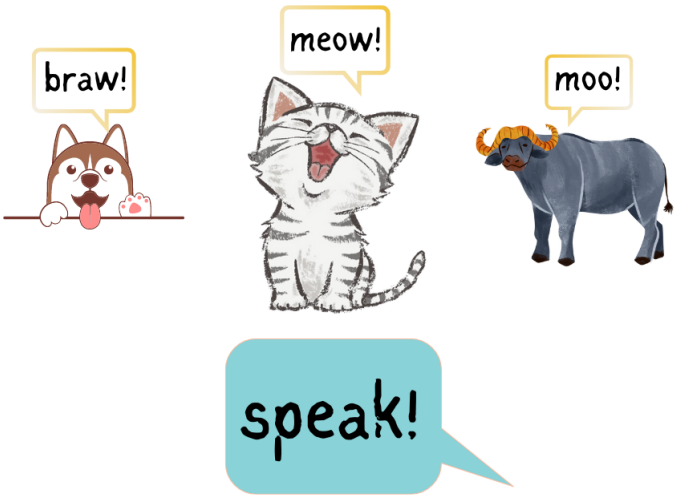
- **Encapsulation:**
  - Technique of bundling data and methods in a class.
  - With encapsulation, data cannot be accessed from outside the class.
  - This protects the data from accidental modification.



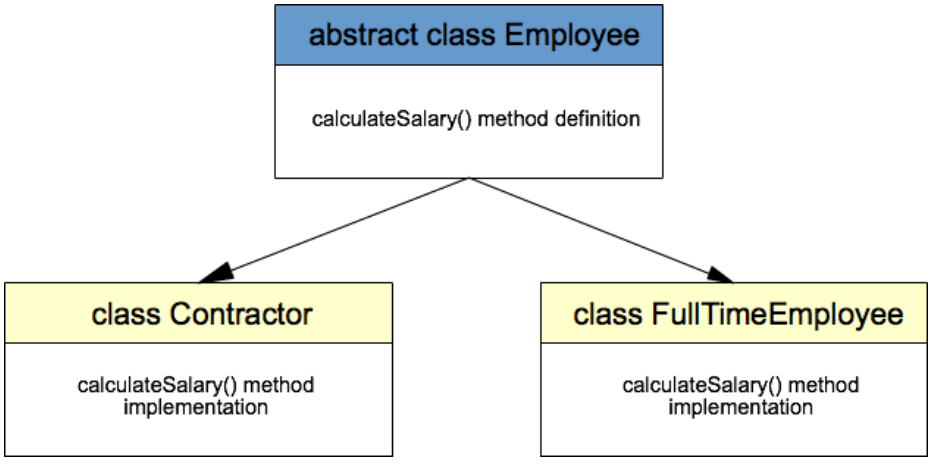
- **Inheritance:**
  - Inheritance is one class acquires the properties (methods and fields) of another class.
  - It allows code reusability and saves time and effort.



- **Polymorphism:**
  - Polymorphism is ability of objects of different classes to be treated as if they belong to a common superclass.
  - Polymorphism allows methods to be written that can work with objects of many different classes, as long as they share a common interface.
  - This enables code to be more flexible and adaptable to changing requirements.



- **Abstraction:**
  - Abstraction hides implementation details.
  - It shows only the essential features of an object.



2.2.2 main() method

- **main()** serves as the entry point for a Java program.
- When a Java program is executed, the JVM starts by looking for the main() method in the class specified in the command line arguments, and then executes the code inside it.

Syntax:

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

- At runtime, JVM always searches for main method with the above prototype:
  - **public:** To call main() from anywhere
  - **static:** without existing object also, JVM has to call this method
  - **void:** main() method wont return anything to JVM
  - **main:** This is the name which is configured inside JVM
  - **String[] args:** command line argument

Note:

The main() syntax is very strict and if we perform any change then we will get runtime error from JVM saying "NoSuchMethodError: main".

- Changes allowed in main():
  - Order of modifier can be changed:
    Eg: static public void main(String[] args)
  - The command line argument's string array can have different syntax:
    Eg: public static void main(String args[])
  - Identifier of the string array can change:
    Eg: public static void main(String name[])
  - String array can be taken as var_arg parameter
    Eg: public static void main(String... args)

- main() method can be declared with following modifiers:
  - final
  - synchroised
  - strictfop

Eg:

New.java

```
class New {
        static final synchronized strictfp public void
main(String... name){
            System.out.println("Valid main method");
    }
}
```

- There can be multiple main() methods (i.e main() method over-loading is possible!). However JVm will always call String[] argument main method only.

New.java

```
class New {
    public static void main(String[] args){
        System.out.println("Starting");
    }     public static void main(int[] args){
        System.out.println("Sample 2");
    } }
```

Output:

```
Starting
```

- **Inheritance:** While executing child class, if child does not contain main(), then parent class main() will be executed.

New.java

```
class New {
    public static void main(String[] args){
        System.out.println("Starting");
    }
}
class C extends New {}
```

Command:

```
$ javac New.java <- Creates C.class New.clas New.java
$ java New
Starting
$ java C
Starting
```

- **Method hiding**: Child class can override parent class's main(). This is not method overriding but it is method hiding.

  Eg:

New.java

```
class New {
    public static void main(String[] args){
        System.out.println("Starting New");
    }
}
class C extends New {
    public static void main(String[] args){
        System.out.println("Starting C");
    }
}
```

Command:

```
$ javac New.java <- Creates C.class New.clas New.java
$ java New
Starting New
$ java C
Starting C
```

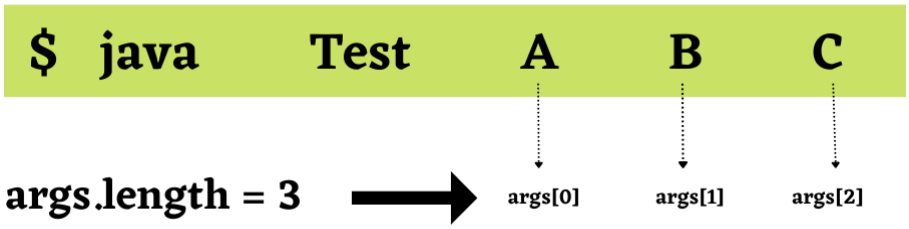
Note:

- Whether class contains main() method or not, and whether main() method is declared according to requirement or not, **these things are won't be checked by compiler**.
- At runtime, **JVM is responsible to check these things**.
- If JVM unable to find main() method, then will throw runtime exception!

2.2.3 Command-line argument

- Command line arguments are values passed to Java program when it is run from the command line.
- With these command line arguments, JVM creates an array and pass it to main().
- Command line arguments can be accessed using the args parameter of the main().
- Args parameter is an array of String objects.
- You can customise behaviour of main() using command-line argument:

```
public static void main(String[] args) // ← Here, String[] args contains command line args
```



- Command line argument are always String[]

New.java

```
class New {
    public static void main(String... args) {
        for(int i = 0; i < args.length; i++) {
        System.out.println(args[i]);
        }
    }
}
```

Command:

```
$ javac New.java
$ java New 1 23 3
1
23
3
```

- Command line arguments are separated by space. To give one argument with space character, using "" :

Command:

```
$ java New "Note Book"
```

Small steps every day.....

3. Data types in Java

3.1 Getting started with data type

In this section, you are going to learn:

1. **Java is strongly typed**
2. **Types of data types**
3. **Integer data type in detail**
4. **Floating-point data type in detail**
5. **Character data type in detail**
6. **Boolean data type in detail**

3.1.1 Java is strongly typed

This means:

- Every variable has a type and every type is strictly defined
- All assignments are checked for type compatibility
- There are no automatic conversions of conflicting types
- Compiler checks all variables to ensure that the types are compatible.
- Any type mismatches errors must be corrected before the compiler will finish compiling the class.

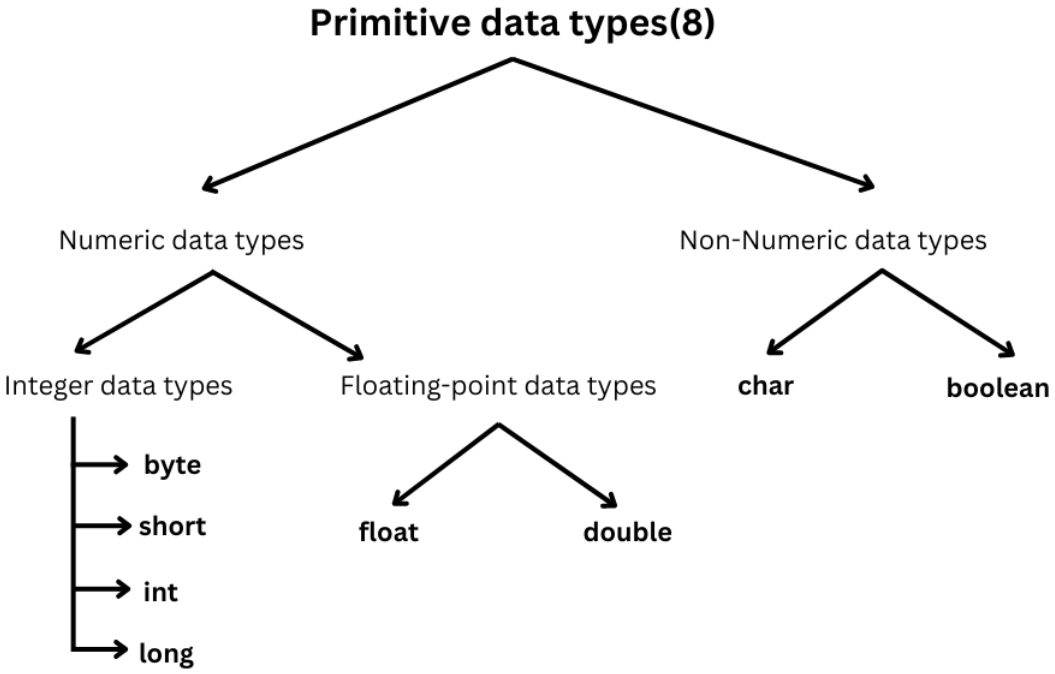
3.1.2 Types of data types

There are two types of data types in Java:

- **Primitive data types**: Includes boolean, char, byte, short, int, long, float and double.
- **Reference data types**:
  - These are not predefined by the language.
  - They are instead created by the programmer using class definitions.
  - Examples of reference data types include:
    * Classes
    * Interfaces
    * Arrays
    * Strings
    * Enumerations

We shall see primitive data type in detail in this chapter.

3.2 Integer



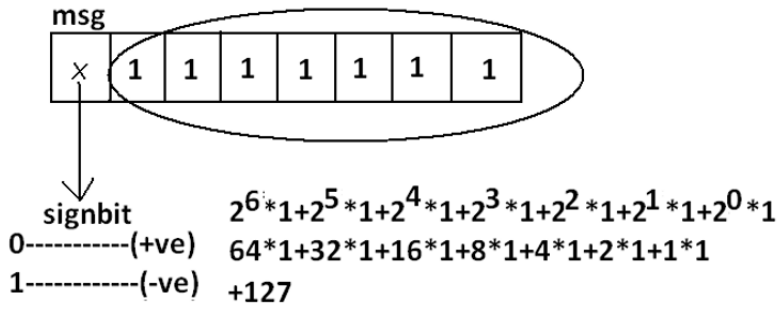
There are 4 types of integer in Java:

- byte
- short
- int
- long

3.2.1 byte

- byte keyword is an 8-bit signed integer.

Size	**1 byte (8 bits)**
MAX_VALUE	**+127**
MIN_VALUE	**-128**
Range	**-128 to 127**



- Left most bit (also called **m**ost **s**ignificant **b**it) is sign bit , where
  - 0 is positive number
  - 1 is negative number

Valid bytes values

```
byte b=10;
byte b=127;
byte b=-120;
```

Invalid bytes values

```
byte b=10.5;
byte b=true;
byte b="lava";
```

- **Where is byte used?**
  - **Reading and writing binary data**
  - **Image processing**
  - **Audio processing**
  - **Network programming**

3.2.2 short

- Least frequently used data type.
- short keyword is 16-bit signed integer.

Size	**2 byte (16 bits)**
MAX_VALUE	32767
MIN_VALUE	-32768
Range	**-2^{15} to $2^{15}-1$**

- **Where is short data type used?**
  - Short data type used for 16 bit processor like 8085.

Valid short values

```
short s = 32767;
short s = -32767;
```

Invalid short values

```
short s=10.5;
short s=true;
```

3.2.3 int

- Mostly commonly used data type is int.
- int keyword is 32-bit signed integer.

Size	**4 byte (32 bits)**
MAX_VALUE	2147483647
MIN_VALUE	-2147483648
Range	**-2^{31} to $2^{31}-1$**

Valid int values

```
int x = 2147483647;
int x = -90;
```

Invalid int values

```
int b=10.5;
int b=true;
int b="lava";
```

3.2.4 long

- long keyword is 64-bit signed integer.

Size	**8 byte (64 bits)**
MAX_VALUE	2^{63}-1
MIN_VALUE	-2^{63}
Range	**-2^{63}** to **2^{63}-1**

- **Where is long data type used?**
  - Eg 1: Amount of distance travelled by light in 1,000 days. To hold this value integer is not enough. Hence, long is used. long l = 1,26,000 X 60 X 60 X 24 X 1000 miles.
  - Eg 2: The number of characters present in a big file may exceed int range. Hence, the return type of length() is long but not integer.

Code:

```
long l = f.length()
```

Long literals

- **long** data type can be suffixed with "l" or "L".
- Below are valid **long** data type:

Code:

```
long l = 10L; ✓
long b = 10; ✓
```

- However, below declaration will result in error:

Code:

```
int x = 10L; ✗
```

3.2.5 Integer literals

- For integral datatypes like byte, short, int & long, we can specify literal value in the following base:
  - **Decimal literal (base-10)**:
    - Allowed digits are 0-9

Code:
```
int x = 10;
```

  - **Binary literal (base-2)**:
    - From Java 1.7 version, integral literal can be represented as binary value.
    - Allowed digits are 0 and 1
    - Literal value should be pre-fixed with "0B" or "0b"

Code:
```
int x = 0B10;
int y = 0b10101;
```

  - **Octal literal (base-8)**:
    - Allowed digits are 0-7
    - Literal value should be pre-fixed with 0

Code:
```
int x = 017;
```

  - **Hexadecimal literal (base-16)**:
    - Allowed digits are 0-9, a-f or A-F
    - Literal value should be pre-fixed with "0X" or "0x"

Code:
```
int x = 0X13aA;
int x = 0x45Fe;
```

  - **Usage of _ in numeric literal**:
    - From Java 1.7 version, we can use "_" in middle of big numbers to increase integer's readability.

* At the time of compilation, these "_" symbols will be removed automatically.

Code:

```
int x = 78_32_34_23;
int y = 0Xaa_ff_11;
int z = 034_12_10;
int a = 0B11__00_10_11;
```

* "_" symbol cannot be used in the starting or end of integer. Below are **invalid** declarations:

Code:

```
int x = _78_32_34_23;
int y = 0Xaa_ff_11_;
```

Program to convert octal and hexadecimal form of integer to decimal form:

test.java

```
package Starter;
class test
{
    public static void main (String[] args)
   {
      int x = 10;
      int y = 061;
      int z = 0x9a;
      System.out.println(x+","+y+","+z);
   }
}
```

Output:

```
10,49,154
```

Output Explaination:

In Java, integeral literals are always represents in decimal literals forms:

- Octal to decimal form:

$$(61)_8 = (?)_{10}$$

$$6 \times 8^1 + 1 \times 8^0 = 49$$

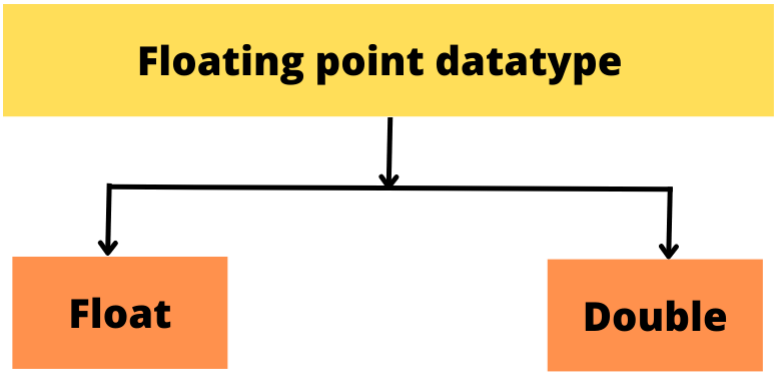
- Hexadecimal to decimal form:

$$(9a)_{16} = (?)_{10}$$

$$9 \times 16^1 + 10 \times 16^0 = 154$$

3.3 Floating-point

Floating-point numbers, also known as real numbers. There are two kinds of floating-point type:



3.3.1 Float

- Represent **single-precision** numbers (upto 7 decimal digits)

Size	**4 byte (32 bits)**
MAX_VALUE	3.4e38
MIN_VALUE	-3.4e38
Range	-3.4e38 to 3.4e38

3.3.2 Double

- Represent **double-precision** numbers (upto 16 decimal digits)

Size	**8 byte (64 bits)**
MAX_VALUE	1.7e+308
MIN_VALUE	1.7e-308
Range	1.7e-308 to 1.7e+308

3.3.3 Floating-point literals

- By default, floating-point numbers are represented in double form.
- So below declaration will result in error:

Code:

```
float f = 1.0 ✗
```

- Correct way to represent float data type is by suffixing **"F"** or **"f"** to the floating-point number as shown:

Code:

```
float f = 1.6F; ✓
float f = 7.8f; ✓
```

- Double data type can be represented using suffix **"D"** or **"d"** or no suffix as below:

Code:

```
double a = 12.67; ✓
double b = 13.7d; ✓
double c = 123.456D; ✓
double d = 0123.456; ✓
double e = 0789.9; ✓
```

- Floating-point literal are only in decimal form, not in octal and hexa decimal forms. Below are **invalid** declarations:

Code:

```
float a = 045.8; ✗
float b = 0X56.9; ✗
double c = 0X56.9; ✗
```

- We can assign integral literal directly to floating-point variables like double and float. Below are valid declarations:

Code:

```
double a = 0456; ✓
double b = 0XFace; ✓
double c = 10; ✓
float a = 0456; ✓
float b = 0XFace; ✓
float c = 10; ✓
```

- **Expontential format:** This is scientific notation to represent very large or small floating-point values. Use the letter “e” or “E” to indicate the exponent:

Code:

```
double a = 1.2e3; ✓
float b = 1.3e4F; ✓
```

- **Hexadecimal floating-point literals:** You can represent double and float in hexadecimal form using the letter “p” or “P”:

Code:

```
double d = 0x12.2P2; ✓
float e = 0x12.2P2f; ✓
```

- **Usage of _ in floating literal**:
  - From Java 1.7 version, we can use "_" in middle of big numbers to increase floating-point’s readability.
  - At the time of compilation, these "_" symbols will be removed automatically.
  - Eg:

Code:

```
float x = 78_3.2_34_23f; ✓
double y = 12_45_23__23_2323.90; ✓
```

- "_" symbol cannot be used in the starting or end of integer or decimal point. Below are **invalid** declarations:
- Eg:

Code:

```
float x = 78_3.2_34_23f_; ✓
double y = _12_45_23__23_2323.90; ✓
double z = 12_45_2_.3__23_2323.90; ✓
```

3.4 Character

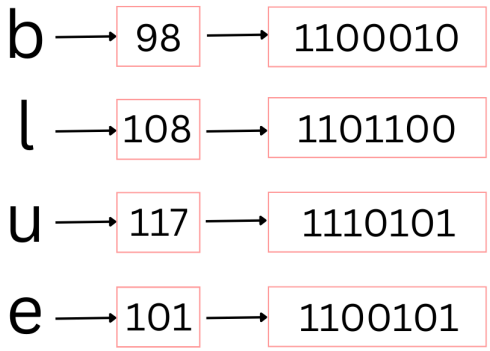
- In order to understand char data type, we need to understand what is "ASCII" and "UNICODE"

3.4.1 What is ascii & unicode?

- **ASCII (American Standard Code for Information Interchange)**:
  - Is a character encoding system that **represents text in computers**.
  - It uses a 7-bit code to represent 128 characters, including the letters of the English alphabet, digits, punctuation marks, and some control codes.

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]



- **Unicode**:
  - It is a character encoding standard designed to support the representation of **all the world's languages**.

3.4.2 char datatype

- Java **uses unicode** to represent **char datatype**.
- There are **no negative chars**.
- Character is represented in **single quotes**.

Size	**2 byte (16 bits)**
MAX_VALUE	65,535
MIN_VALUE	0
Range	0 to 65,535

Eg:

Code:

```
class New {
    public static void main(String[] args) {
        char a = 88;
        char b = 'x';
        System.out.println(a);
        System.out.println(b);
    }
}
```

3.4.3 Character literals

- Char literal can be represented as **single character within single quotes**.

Code:

```
char ch='a'; ✓
```

- Below char literal declaractions will **result into compile time errors**:

Code:

```
char ch = "a"; ✗
char ch = a; ✗
char ch = 'ab'; ✗
```

- Char literal can also be **represented as integral literal** which represents the unicode value of character.
  The unicode value can be specified in decimal, octal and hexa decimal form.

Code:

```
char ch1 = 97; ✓
char ch2 = 0xFace; ✓
char ch3 = 0777; ✓
char ch = 65535; ✓
```

  Note that allowed range is **0 to 65535**.

- Char literal can also be represented in **unicode representation** by using "\uXXXX" syntax where "XXXX" is 4 digit hexa decimal number.

Code:

```
char ch1 = '\u0052'; ✓
char ch2 = '\u0932'; ✓
char ch3 = \uface; ✓
```

- Char literal can also **represent escape sequence characters**.

Code:

```
char ch1 = '\n'; ✓
char ch2 = '\t'; ✓
```

- Below are some more example of **invalid** char declarations:

Code:

```
char ch1 = 65536; ✗
char ch2 = 0XBear; ✗
char ch3 = '\m'; ✗
char ch4 = '\iface'; ✗
```

3.4.4 Escape character

A character preceded by a backslash (\) is an escape sequence and has special meaning to the compiler.
The following table shows the Java escape sequences:

Escape Character	Decimal Point
\n	New line
\t	Horizontal Tab
\r	Carriage return (Move to first character in next line)
\b	Back Space
\f	form feed
\’	single quote
\"	double quote
\\	Back Slash

Eg:

Code:

```
System.out.println("This is line one \n And this is line two");
System.out.println("This is \t tab space");
System.out.println("Sunflower \r Forest");
System.out.println("Sunflower \f Forest \f ground");
System.out.println("C:\\lavatech_technology");
System.out.println("Sunflower\’s Forest");
```

3.5 Boolean

- Boolean is datatype for logical values.
- Boolean is return by all relational operators.

Size	**1 byte (8 bits)**. The actual memory usage may depend on JVM implementation.
Value	true, false

Valid boolean values

```
boolean b = true;
boolean flag = false;
```

Invalid boolean values

```
boolean b = True;
boolean b = "True";
boolean b = 0;
```

Note:

- The values of true and false do not convert into any numerical representation.
- The true literal in Java does not equal 1, nor does the false literal equal 0.

Like C and C++, below code will not result in interpreting 0 and 1 as boolean and result in error

```
int x = 0;
if (x)
{
   System.out.println("Hello");
}
else
{
   System.out.println("Hi");
}
```

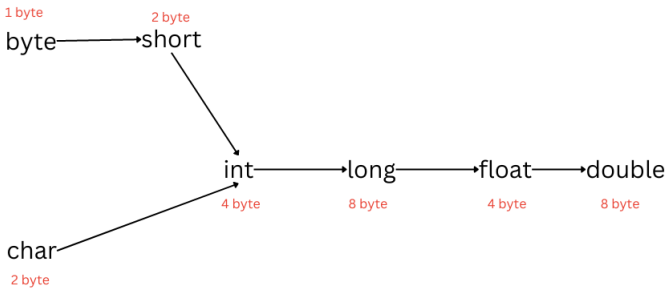
```
while(1)
{
   System.out.println("Hello") ;
}
```

Compile-time error: incompatible types
found: int
required: boolean

3.6 Type conversion

Converting one primitive data type into another is known as type conversion. There are two types of type conversions:

- **Implicit type casting or widening:**
  - **Converting a lower datatype to a higher datatype** is known as widening or up-casting.
  - Compiler is responsible to perform implicity type casting.
  - There is no loss of information in this type casting.
  - The following are various possible conversions:



Eg:

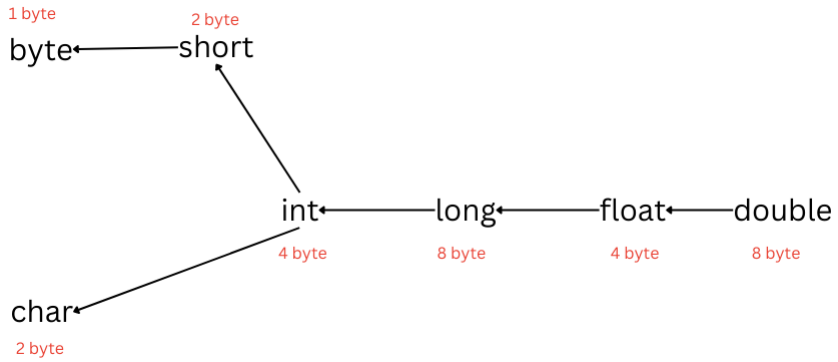
Code:

```
int x = 'a';
System.out.println(x); // output: 97

double d = 10;
System.out.println(d); // output: 10.0
```

Note:

- **Long value can be assigned to float variable** because both are following different memory representations internally.

- **Explicit type casting or narrowing:**
  - Converting a higher datatype to a lower datatype is known as narrowing or down casting.
  - Programmer is responsible to perform explicit type casting.
  - Loss of information is possible.
  - The following are various possible conversions:



  - Except for boolean, all datatypes can be type-cast to other primitive data-type.
  - Type-cast operator for each primitive datatype:

Data type	Type-case operator	Example
byte	(byte)	double d = 130.4; byte x = (byte) d;
short	(short)	double d = 130.4; short x = (short) d;
int	(int)	double d = 130.4; int x = (int) d;
long	(long)	double d = 130.4; long x = (long) d;
float	(float)	double d = 130.4; float x = (float) d;
double	(double)	float d = 130.4; double x = (double) d;
char	(char)	double d = 130.4; char x = (char) d;

Eg 1:

Code:

```
int x = 130;
//byte b = x; ✗
byte b = (byte) x; ✓
System.out.println(b); // output: -126
```

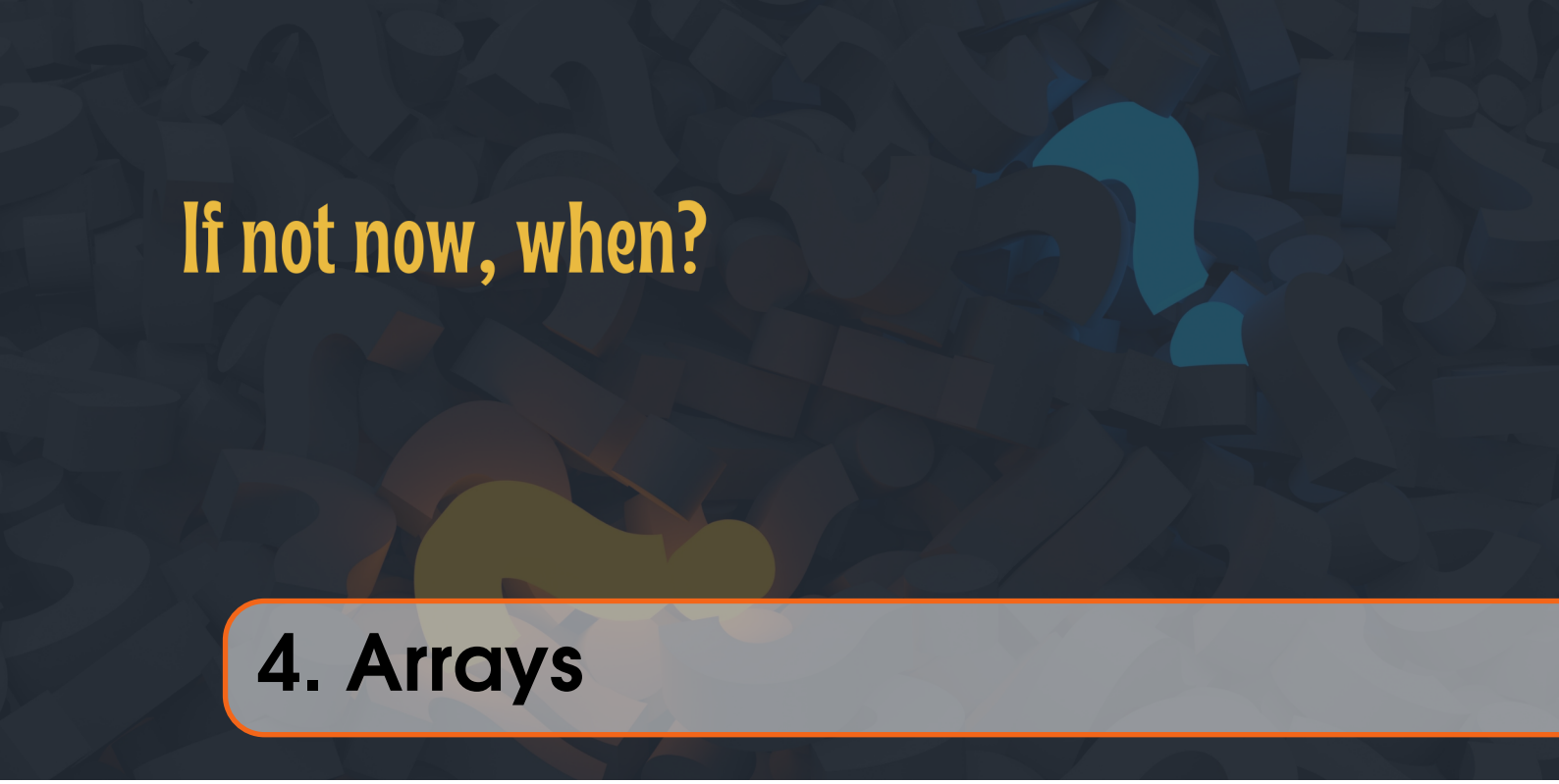
Output explaination:

- Integer is 32-bit in size & byte is 8-bit in size.
- 32-bit binary representation of 130 is 0000000...10000010.
- To convert integer to byte datatype, mean downsize decimal representation of 130 to fit in 8 bit.
- Which means the binary number **0000000...10000010** is down-sized to **10000010**
- Note that left-most bit is "1" and hence number is now represented as 2's complement.
- 2's complement of **10000010** is **11111101+1 => 11111110 => -126**

Eg 2: If we assign floating point values to integral types, by explicit type casting, the digits after decimal point will be lost.

Code:

```
double d = 130.456;
int x = (int) d;
System.out.println(x); // output: 130

byte b = (byte) d;
System.out.println(b); // output: -126
```



4. Arrays

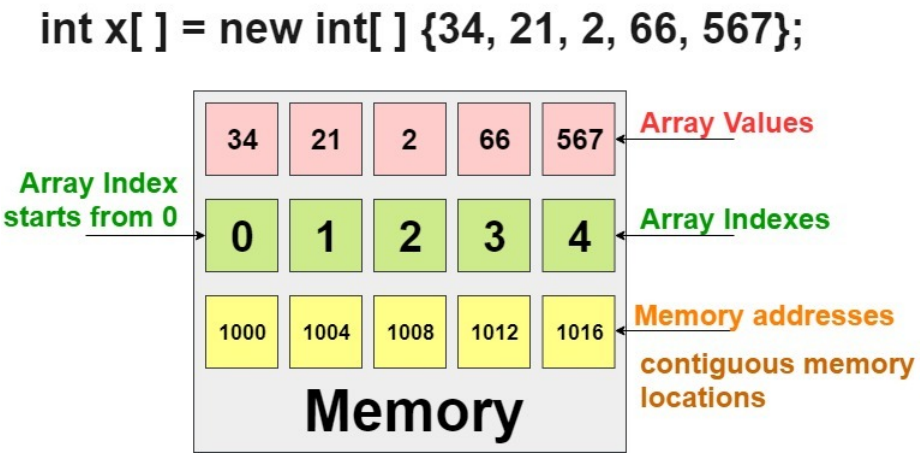
4.1 Arrays in detail

In this section, you are going to learn:

Selection Statements	Iterative Statements	Transfer Statements
• if..else • switch()	• while() • do-while() • for() • for-each loop (Java 1.5)	• break • continue • return • try..catch.finally • assert

4.1.1 Array introduction

An array is **indexed collection of fixed number of homogeneous data elements**.



Advantage:

- Array can represent huge number of values using single variable that will improve readability of code.

Disadvantage:

- Array are fixed in size.
- Once an array is created, they cannot be increased or decreased.
- Array size need to be mentioned in advance, which is not always possible.

4.1.2 Array declaration

- **One dimensional array declaration**:

Syntax:

int[] x; (Recommended as name of variable is clearly separated from type)
int []x;
int x[];

Note:

Array declaration **cannot define size** of array.

Code:

```
int[6] x; ✗
```

- **Two-dimensional array declaration**:

Syntax:

int[][] x; (Recommended)
int [][]x;
int x[][];
int[] []x;

- **3-dimensional array declaration:**

Syntax:

int[][][] x; (Recommended)
int [][][]x;
int x[][][];
int[] [][]x;
int[] x[][];
int[] []x[];

- **More combinations**:
  - Declaring variable **"a"** and **"b"** with **1** dimension:

Code:

```
int[]  a,b;
```

  - Declaring variable **"a"** with 2 dimension and variable **"b"** with 1 dimension

Code:

```
int[]  a[],b;
```

  - Declaring variable **"a"** and **"b"** with 2 dimensions.

Code:

```
int[]  a[],b[];
int[]        a[],b;
```

  - Declaring variable **"a"** with 2 dimension and variable **"b"** with 3 dimension:

Code:

```
int[]        []a,b[];
```

Note:

"[]" is allowed only in front of first variable.

Code:

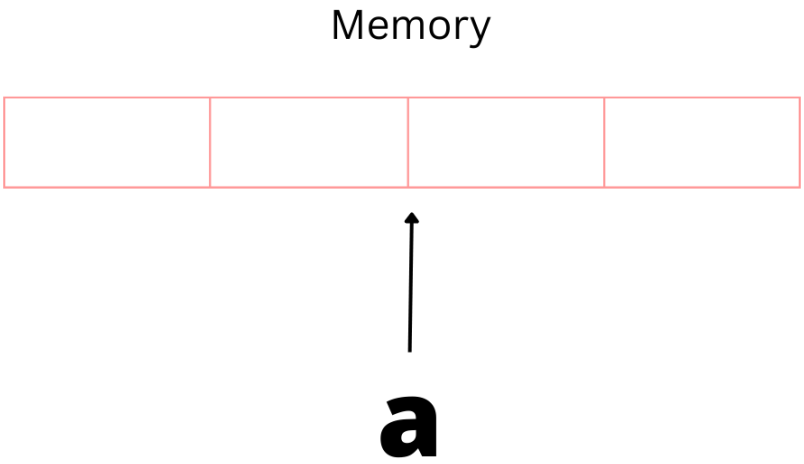
```
int[]      []a,[]b;  ✗
int[]      []a,[]b,[]c;  ✗
```

4.1.3 Array creation

Things to note about array:

- In Java, every array is an Object.
- **"new"** operator is used to create an object.
- Hence, we can create array by using **new** operator.
- **One dimensional array creation**:

```
Syntax:
int[] a = new int[4];
```



Important:

- At the time of array creation, **size should be mentioned compulsorily.**
- An array can be of zero size.

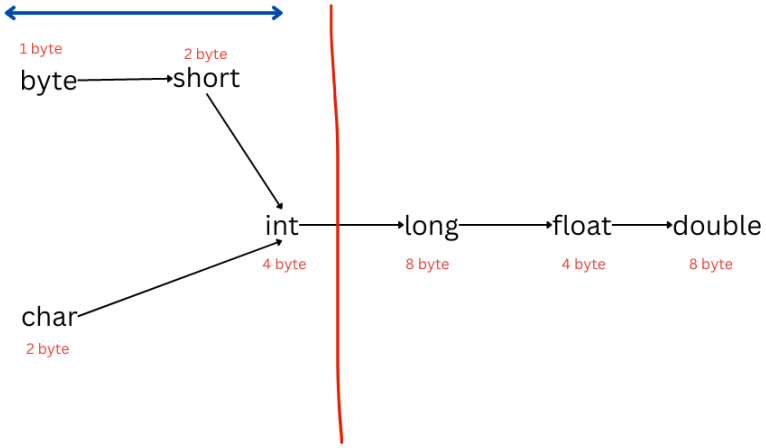
```
int[] x = new int[]; ✗
int[] x = new int[6]; ✓
int[] x = new int[0]; ✓
```

- Java compiler will never throw error for negative size of array. However, Java Virtual Machine will throw runtime error:

NegativeArraySizeException.

```
int[] x = new int[-3]; ✗
```

– Allowed data types for mentioning array size are:
  * integer
  * byte
  * short
  * char




```
int[] x = new int[10]; ✓
int[] x = new int['a']; ✓
byte b = 20;
int[] x = new int[b]; ✓
short s = 30;
int[] x = new int[s]; ✓
```

Below array creation will result in error:

```
int[] x = new int[10l]; ✗
int[] x = new int[3.5]; ✗
```

– Maximum size of array can be 2147483647:

```
int[] x = new int[2147483647]; ✓
int[] x = new int[2147483648]; ✗
```

– For every array type, corresponding classes are available and these classes are part of Java language and not available to the programmer level.

Array type	**Corresponding class name**
int[]	[I
int[][]	[[I
double[]	[D
short[]	[S
byte[]	[B
boolean[]	[Z

Eg: You can find name of class for different array type:

```
int[] a = new int[3];
System.out.println(a.getClass().getName());
```

Output:

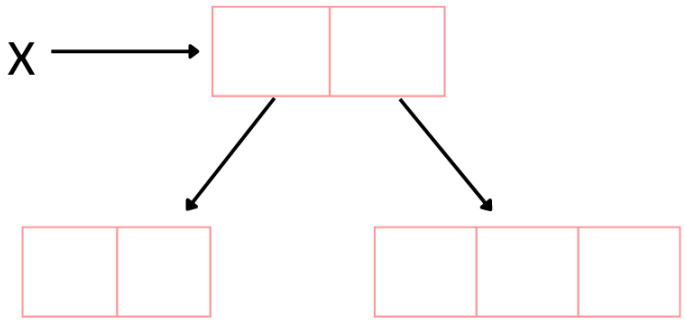
```
[I
```

- **Two-dimensional array creation**:
  - In Java, two dimensional array is not implmeneted using matrix approach.
  - Array of arrays approach is followed for multi-dimensional array creation.
  - Advantage of array of arrays approach is improved memory utilisation.

There are different ways of creating two-dimensional array.

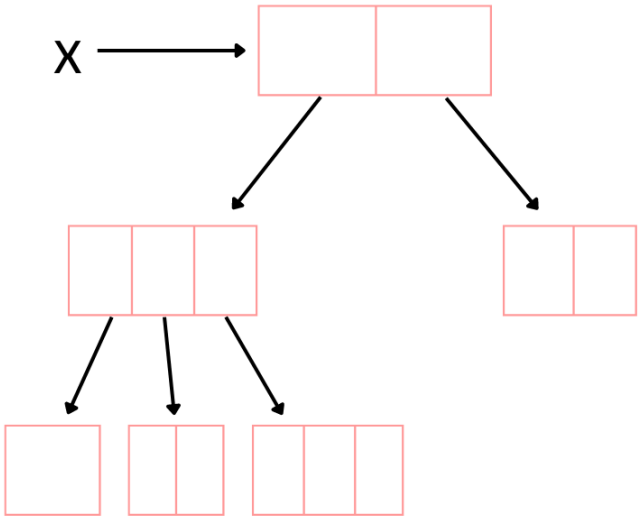
- **Base size**: In this we specifiy the size of first dimension at the time of array creation.

```
int[][] x = new int[2][];
x[0] = new int[2];
x[1] = new int[3];
```



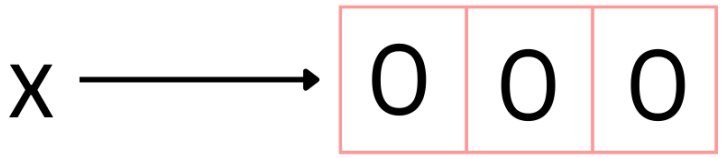
- Three-dimensional array creation:

```
int[][][] x = new int[2][][];
x[0] = new int[3][];
x[0][0] = new int[1];
x[0][1] = new int[2];
x[0][2] = new int[3];
x[1] = new int[2][2];
```



4.1.4 Array initialisation

- **One dimensional array**:
  Once we create an array, every array element is by default initialized with default values.



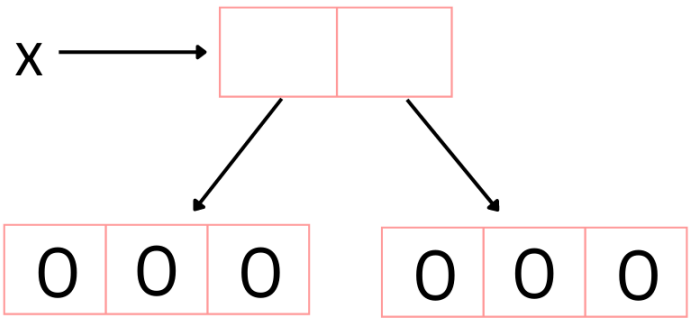

```
int[] a = new int[3];
System.out.println(a);
System.out.println(a[0]);
```

Output:

```
[I@422a8473
0
```

Whenever we are trying to print any reference variable, internally two string method will be called, which is implemented by default to return the string in the following form:
class_name@hexadecimal_form

- **Two-dimensional array**:
  Example 1:

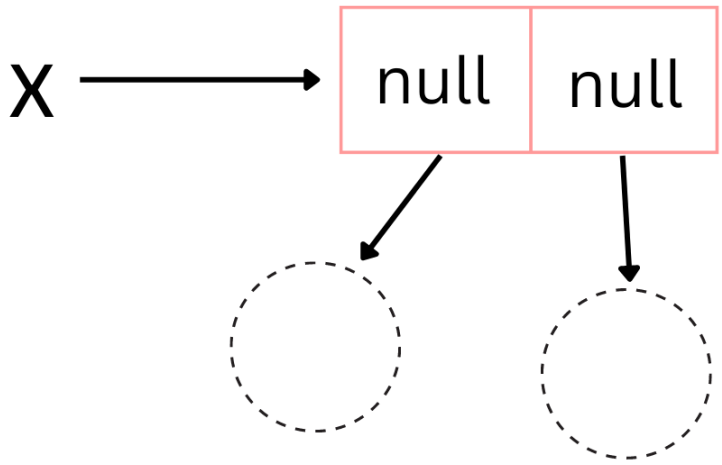


```
int[][] a = new int[2][3];
System.out.println(a);
System.out.println(a[0]);
System.out.println(a[0][0]);
```

Output:

```
[[I@5a39699c
[I@129a8472
0
```

Example 2:

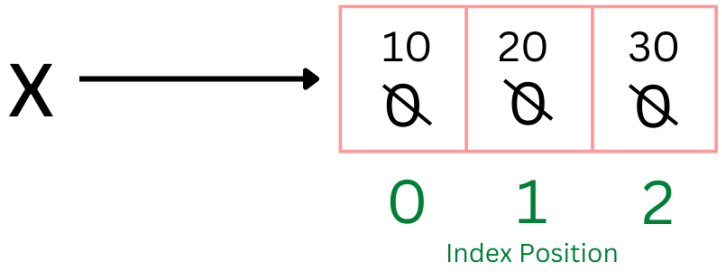



```
int[][] a = new int[2][];
System.out.println(a);
System.out.println(a[0]);
System.out.println(a[0][0]);
```

Output:

```
[[I@5a39699c
null
Exception in thread "main" java.lang.NullPointerException:
```

- **Over-riding array value:**
  Once we create an array, every array element by default initialised with default values.
  We can over-ride default values with custom values.




```
int[] a = new int[3];
a[0]=10;
a[1]=20;
a[2]=30;
System.out.println(a[0]);
System.out.println(a[1]);
System.out.println(a[2]);
```

Output:

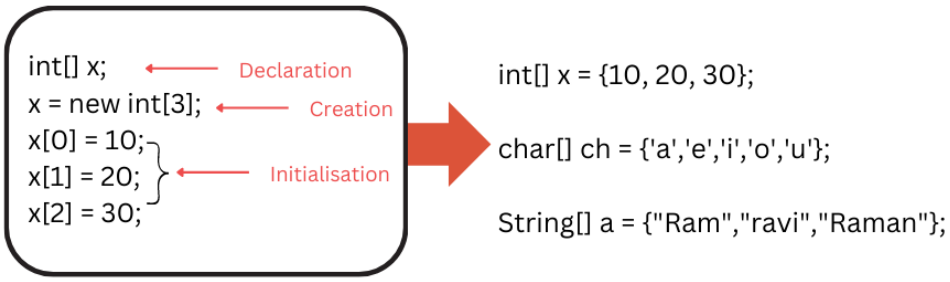
```
10
20
30
```

Note: Trying to access array element with out of range index (either positive or negative integer value) will result in runtime exception: **"ArrayINdexOutOfBoundException"**

4.1.5 Array declaration, creation and initialisation in one line

- **One dimensional array**:

  We can declare, create and initialise an array in a single line (shorcut representation):



Code:

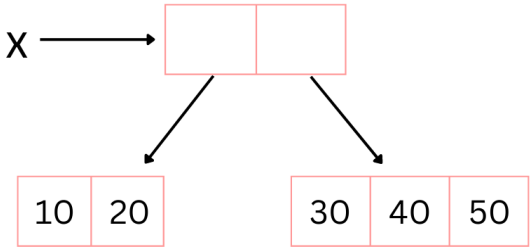
```
int[] x = {10,20,30};
char[] ch = {'a','e','i','o','u'};
String[] a = {"Ram" , "Ravi"};
```

You can declare the array & provide it's value as shown below:

Code:

```
int[] x;
x = {10,20,30};
```

- **Multi-dimensional array**:



Code:

```
int[] x = { { 10, 20 }, { 30, 40, 50 } };
```

4.1.6 length variable

length variable:

- length is final variable applicable for arrays.
- length variable is used to display size of an array.
- Value returned by length is fixed as array once created cannot change it's size.

Code:

```
int[] x = new int[6];
System.out.println(x.length); ✓
```

Output:

```
6
```

- length variable is **not** applicable on string objects.

Code:

```
String s="lavatech";
System.out.println(s.length); ✗
```

- In multi-dimensional arrays, length variable represents only base size, but not total size.

Code:

```
int[][] x = new int[6][3];
System.out.println(x.length);
```

Output:

```
6
```

length():

- length() is present in String class.
- length() method is final variable applicable for string objects.

- It returns number of characters present in the string.

Code:

```
String  s="lavatech";
System.out.println(s.length()); ✓
```

Output:

```
8
```

- length variable is applicable for arrays, but not for string objects.
- length() is applicable for string objects, but not for arrays.

  Example:

Code:

```
String[]  s=  {"A","AA","AAA"};
System.out.println(s.length); ✓
System.out.println(s.length()); ✗
System.out.println(s[0].length); ✗
System.out.println(s[0].length()); ✓
```

Output:

```
3
error
error
1
```

- There is no direct way to find total length of multi-dimensional array.

  Total length of multi-dimensional array can be found as follows:

Code:

```
int[][]  x  =  new  int[3][3];
System.out.println(x.length);
System.out.println(x[0].length+x[1].length+x[2].length);
```

Output:

```
3
9
```

4.1.7 Anonymous Arrays

- Anonymous arrays are nameless arrays.
- These arrays are used for instant one-time purpose.

Syntax:

Single dimension array: **new datatype[]{}**
Multi-dimension array: **new datatype[][]{{},{}}**

- While creating anonymous arrays, you cannot mention it's size:

Code:

```
new int[3]{10,20,30} ✗
new int[]{10,20,30} ✓
new int[][]{{10,20,30},{40,50,60}} ✓
```

- In below example, main() is calling sum() using an anonymous arrays:

Test.java

```
{
    public static void main (String[] args)
    {
        sum(new int[]{10,20,30,40,50});
    }
    public static void sum(int[] x)
    {
        int total = 0;
        for(int x1 :x)
        {
            total = total+x1;
        }
        System.out.println("The sum is : "+total);
```

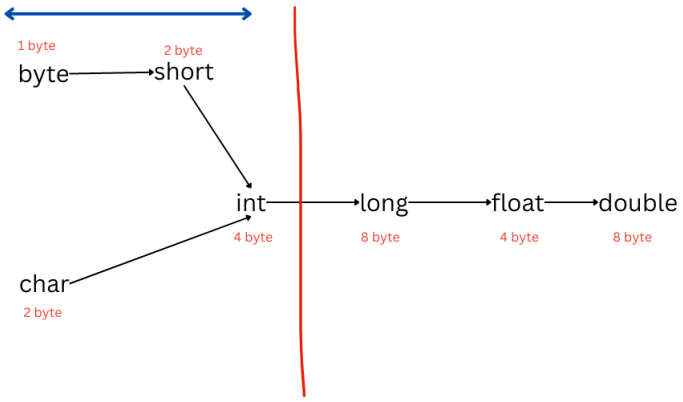
```
        }
}
```

Output:

```
The sum is : 150
```

4.1.8 Array element assignments

In case of **primitive type arrays**, as array elements, you can provide any type which can be **implicitly promoted to declared type**. Example:



Code:

```
int[] x = new int[5];
x[0] = 10;
x[1] = 'a';
byte b = 20;
x[2] = b;
short s = 30;
x[3] = s;
x[4] = 10L; ✗
```

Similary, in case of float type arrays, the allowed datatypes are:

- byte
- short
- char
- int
- long
- float

4.1.9 Array variable assignments

- Data type level promotion are not applicable at array level.
- Eg: Char datatype can be promoted to int type. Whereas, char array cannot be promoted to int array.

Code:

```
int[] x = {10,20,30,40};
char[] ch = {'a','b','c','d'};
int[] b = x; ✓
int[] c = ch; ✗
```

Which of the following promotions will be performed automatically:

Conversion	Answer
char → int	✓
char[] → int[]	✗
int → double	✓
int[] → double[]	✗
float → int	✗
float[] → int[]	✗

Don't fear failure.

Not failure, but low aim is the crime.

5. Operators

5.1 Operators in Java

In this section, you are going to learn:

1. **Arithematic Operator**
2. **String Concatenation Operator**
3. **Increment/Decrement Operator**
4. **Relational Operator**
5. **Equality Operator**
6. **Bitwise Operator**
7. **Boolean complement Operator**
8. **Short circuit Operator**
9. **Type cast Operator**
10. **Assignment Operator**
11. **Conditional Operator**
12. **new Operator**
13. **[] Operator**

14.

5.1.1 Arithematic Operator

Operator	Example

Addition(+)	`int a = 5;`
	`int b = 10;`
	`int c = a + b; // c will be 15`

Subtraction(-)	`int a = 10;`
	`int b = 5;`
	`int c = a - b; // c will be 5`

Multiplication(*)	`int a = 2;`
	`int b = 3;`
	`int c = a * b; // c will be 6`

Modulus (%)	`int a = 10; int b = 3; int c = a % b; // c will be 1`

Division(/)	int a = 10;
	int b = 3;
	int c = a / b; // c will be 3 (the remainder is discarded)
	
	double d = 10.0;
	double e = 3.0;
	double f = d / e; // f will be 3.33333

Important Points:

- **Implicit type casting:** If we apply any arithmetic operator between 2 variables "a" and "b", the result type is always:

  maximum(int, type of a, type of b)

  Using above formula,

  - Byte + byte = int
  - Byte + short = int
  - Short + short = int
  - Byte + long = long
  - Long + double = double
  - Float + long = float
  - Char + char = int
  - Char + double = double

Eg:

Code:

```
System.out.println('a'+'b'); // output: 195
System.out.println('a'+3.29); // output: 100.29
```

- **Infinity**:
  - In integral arithmetic **(byte short int long), infinity cannot be represented** and JVM will return runtime error.

    Code:

    ```
    System.out.println(10/0); ✗
    ```

  - But in **floating point arithmetic (float, double), infinity can be represented**. For this, Float and Double classes contains below 2 constants:
    - POSITIVE_INFINITY;
    - NEGATIVE_INFINITY;

    Code:

    ```
    System.out.println(10/0.0); // output: Infinity
    System.out.println(-10/0.0); // output: -Infinity
    ```

- **NaN (not a number):**
  - In **integer arithmetic (byte, short, int, long) , undefined results cannot be represented** and JVM will return runtime error.

    Code:

    ```
    System.out.println(0/0); ✗
    ```

  - But, in floating point arithmetic **(float,double), undefined results can be represented as NaN constant**.

Code:

```
System.out.println(0.0/0); // output: NaN
System.out.println(-0/0.0); // output: NaN
```

5.1.2 String Concatenation Operator

- The only overloaded operator in Java is **"+"** operator.
- It can act as arithmetic addition operator as well as string concatecation operator.

Code:

```
System.out.println(10+20); // output: 30
System.out.println("ab"+"cd"); // output: abcd
```

Note:

Apart from "+", Java does not support **operator overloading**!

- Working of **"+"** with string:
  - If atleast one argument is string type, + operator acts as concatenation operator.
  - If both arguments are number type, + operator acts as arithmetic addition operator.

Eg:

Code:

```
String a = "lavatech";
int b=10, c=20, d=30;
System.out.println(a+b+c+d); // output: lavatech102030
System.out.println(b+c+d+a); // output: 60lavatech
System.out.println(b+c+a+d); // output: 30lavatech30
System.out.println(b+a+c+d); // output: 10lavatech2030
```

5.1.3 Increment/Decrement Operator

- The increment(++) and decrement(–) operators are unary operators.
- They are used to increment or decrement the value of a variable by 1.
- **Increment Operator (++):** Used in two ways:
  - **Prefix (++var)**: Variable is incremented first and then used in the expression.

Code:

```
int a = 5;
int b = ++a; // b will be 6, a will be 6
```

  - **Postfix (var++)**: Variable is used in the expression and then incremented.

Code:

```
int a = 5;
int b = a++; // b will be 5, a will be 6
```

- **Decrement Operator (–):** Works in a similar way and can be used in prefix and postfix forms:

Code:

```
int a = 5;
int b = –a; // b will be 4, a will be 4
int c = a–; // c will be 4, a will be 3
```

Summary:

Expression	Initial value of x	Value of y	Final value of x
y=++x;	10	11	11
y=++x;	10	10	11
y=++x;	10	9	9
y=++x;	10	10	9

Important Points:

- Increment/decrement is **applicable only on variable and not on constant**.

Code:

```
System.out.println(++10); ✗
```

- **Listing** of increment/decrement operators **not allowed**.

Code:

```
int x=10;
int y = ++(++x); ✗
```

- **For final variables**, increment/decrement operators **cannot** be used:

Code:

```
final int x=10;
System.out.println(x++); ✗
```

- Increment/decrement is applicable on all primitive type, **except boolean datatype**:
  - Integer example:

Code:

```
int x=10;
x++;
System.out.println(x); // output: 11
```

  - Character example:

Code:

```
char ch = 'a';
ch++;
System.out.println(ch); // output: 'b'
```

- Boolean example:

Code:

```
boolean b=true;
b++; ✗
```

Difference between “x++” and “x=x+1”

- We know that: If we apply any arithmetic operator between 2 variables “a” and “b”, the result type is always:

```
maximum(int, type of a, type of b)
```

- This is the reason why using "x=x+1" can result in compile-time error:

Code:

```
byte b=10;
b = b+1; ✗
```

- But in case of increment/decrement operators, internal type casting will be performed automatically:

Code:

```
byte b=10;
b++; ✓
```

5.1.4 Relational Operator

- Below are available relational operator in Java:

```
<  ----> less than
<= ----> less than equal to
>  ----> greater than
>= ----> greater than equal to
== ----> equal to
!= ----> not equal to
```

- We can apply relational operator for every primitive datatype, except boolean datatype.
- Eg:

Code:

```
System.out.println(10>20); // output: false
System.out.println('a'>20); // output: false
System.out.println('b'>2.0); // output: false
//System.out.println(true > false); ✗
```

- We can't apply relational operators for object types

Code:

```
System.out.println('lava' > 'lavatech'); ✗
```

- Chaining of relational operators is not allowed.

Code:

```
System.out.println(10>20>30); ✗
```

5.1.5 Equality Operator

- Equality operators can be used for every primitive type including boolean.

Code:

```
System.out.println(10==20); // output: fasle
System.out.println('a' == 'b'); // output: false
System.out.println('a' == 97.0); // output: true
System.out.println(false == false); // output: true
```

- Equality operators can be applied for object types also. For object references, "r1" & "r2", "r1==r2" returns true, if both reference pointing to the same object (reference comparison or address comparison)

Code:

```
Thread t1 = new Thread();
Thread t2 = new Thread();
Thread t3 = t1;
System.out.println(t1 == t2); // output: false
System.out.println(t1 == t3); // output: true
```

There should be some relation between argument types(either child to parent or parent to child or same type). Otherwise, it will result in compile-time error.

Code:

```
Thread t1 = new Thread();
Object o = new Object();
String s = new String("lava");
System.out.println(t1==o) ; // output: false
System.out.println(o==s); // output: false
//System.out.println(s==t1); ✗
```

For any object reference "r": "r==null" ← is always False

Code:

```
String s1 = new String("lava");
System.out.println(s1 == null) ; // output: false

String s2 = new String();
System.out.println(s2==null); // output: false

String s3 = null;
System.out.println(s3==null); // output: true
```

5.1.6 Bitwise Operator

& ---> Bitwise And
| ---> Bitwise Or
^ ---> Bitwise Xor
~ ---> Bitwise Complement
>> ---> Bitwise left shift
<< ---> Bitwise right shift

Bitwise & - If both bits are 1, then only 1 otherwise 0

Sample Code	Output	Explaination

int a=4;
int b=5;
System.out.println(a & b)
	4	100
		101
		———
		100

Bitwise | - If atleast one bit is 1, then only 1 otherwise 0

Sample Code	Output	Explaination

int a=4;
int b=5;
System.out.println(a | b)
	5	100
		101
		———
		101

Bitwise ^ - Also called x-or. If both bits are different, then 1, otherwise 0

Sample Code	Output	Explaination

int a=4;
int b=5;
System.out.println(a ^ b)
	1	100
		101
		—
		001

Bitwise ~ - Bitwise complement operator, 1 becomes 0 and 0 becomes 1

Sample Code	Output	Explaination
int a=4; System.out.println(~ a)	-5	32-bit os represents number with total 32 bits as 000...000100 ~ 000...000100 = 111...111011 Left most bit is sign bit where, 1 is negative and 0 is positive Since left most bit is now 1, number is negative Negative number is represented as 1's complement + 1 ie. 100..000100 + 1 = 100...000101

Bitwise » - Bitwise right shift.

Remove "x" bit from right side and add "x" 0 to left side.

Sample Code	Output	Explaination

int a=10;
int x=2;
System.out.println(a » x)
	2	000...1010 » 2
		000...0010

Bitwise « - Bitwise left shift.

Remove "x" bit from left side and add "x" 0 to right side.

Sample Code	Output	Explaination

int a=10;
int x=2;
System.out.println(a « x)
	40	000...1010 « 2
		000...101000

Note:

- &, | , ^ are applicable for both boolean and integral type.
- » , « are applicable for integral type only.
- ~ applicable for only integral type but not for boolean type.

5.1.7 Boolean complement operator

- The Boolean complement operator (!) is a unary operator that negates the value of a Boolean expression.

Code:

```
boolean a = true;
boolean b = !a; // output: b is false
```

Note:

! operator can be applied only on boolean data-type.

5.1.8 Short circuit Operator

In Java, the short-circuit operators are:

- **&& (logical AND)**: Same as bitwise &
- || **(logical OR)**: Same as bitwise |

& , |	&& , ||
Both arguments should be evaluated always	Second argument evaluation is optional
Relatively performance is low	Relatively performance is high
Applicable for both boolean and integral types	Applicable only for boolean but not for integral types
Eg: For x & y, both x and y will be evaluated. Similarly, For x | y, both x and y will be evaluated.	Eg: For x && y, y will be evaluated if and only if x is true i.e if x is false then y wont be evaluated. Similarly, For x || y, y will be evaluated if and only if x is false i.e if x is true then y wont be evaluated.

Consider below code showing different behaviour of &, &&, |, ||:

Code:

```
int x = 10, y = 15;
if( ++x < 10 & ++y > 15) {
    x++;
}
else {
    y++;
}
System.out.println(x + "..." + y); // output: 11...17
```

Code:

```
int x = 10, y = 15;
if( ++x < 10 | ++y > 15) {
    x++;
}
else {
    y++;
}
System.out.println(x + "..." + y); // output: 12...16
```

Code:

```
int x = 10, y = 15;
if( ++x < 10 && ++y > 15) {
    x++;
}
else {
    y++;
}
System.out.println(x + "..." + y); // output: 11...16
```

Code:

```
int x = 10, y = 15;
if( ++x < 10 || ++y > 15) {
    x++;
}
else {
    y++;
}
System.out.println(x + "..." + y); // output: 12...16
```

5.1.9 Assignment Operator

There are 3 types of assignment operators:

- **Simple:** The assignment operator (=) is used to assign a value to a variable.

Code:
```
int x = 10;
```

- **Chained:**

Code:
```
int a,b,c,d;
a=b=c=d=20;
```

We can't perform chained assignment directly at the time of declaration:

Code:
```
int a=b=c=d=20; ✗

int b,c,d;
int a=b=c=d=20; ✓
```

- **Compound:**
  - Assignment operator can be mixed with other operators.
  - Such type of assignment operators are called compound assignment operators.

Syntax:
```
variable operator= expression;
```

Code:
```
int a = 10;
a += 20;
System.out.println(a) ; // output: 30
```

– In case of compound assignment operator, internal type casting will be performed automatically:

Code:

```
byte b=10;
b = b+1; ✗

byte b = 10;
b++; // output: 11

byte b=10;
b+=1; // output: 11
```

– Below are sample examples of compound assignment operator:

Code:

```
int x = 5; x += 3; // x is 8
x -= 2; // x is 6
x *= 4; // x is 24
x /= 3; // x is 8
x %= 5; // x is 3
x &= 1; // x is 1
x |= 2; // x is 3
x ^= 3; // x is 0
x «= 2; // x is 0
x »= 1; // x is 0
```

5.1.10 Conditional Operator

- The conditional operator (also known as the ternary operator)
- It is **if-else statement in a single line**.

Syntax:

```
condition ? expression1 : expression2
```

- If condition is true, then the expression1 is evaluated else expression2 is evaluated and its value is returned.

Code:

```
int a = 23, b = 30;
System.out.println( (a>b)? "a is greater" : "b is greater");
```

5.1.11 new Operator

- **new** operator is used to create an object.

Syntax:

```
Class obj = new class();
```

Eg:

Code:

```
String name = new String("Ram");
```

5.1.12 [] Operator

- [] operator is used to declare and create arrays.

Code:

```
int[] x = new int[10];
```

5.1.13 Operator Precedence

Unary operators	[] , x++ , x– ++x , –x , ~ , !
Arithematic operators	* , / , % + , -
Shift operators	» , »> , «
Comparison operators	< , <= , > , >= , instanceof
Equality operators	== , !=
Bitwise operators	& , ^ , |
Short circuit operators	&& , ||
Conditional operator	?:
Assignment operators	= , += , -= , *=

Evaluation order of operands:

- In java, we have only operator precedence and no operands precedence.
- Before applying any operator, all operands will be evaluated from left to right.
- Eg:

Code:

```
System.out.println(1+2*3/4+5*6);  //  output:  32
```

Output explaination:

  - 1+2*3/4+5*6
  - 1+6/4+5*6
  - 1+1+5*6
  - 1+1+30
  - 32

Don't fear failure.
Not failure, but low aim is the crime.

6. Flow Control

6.1 Flow control statement

In this section, you are going to learn:

Selection Statements	Iterative Statements	Transfer Statements
• if..else • switch()	• while() • do-while() • for() • for-each loop (Java 1.5)	• break • continue • return • try..catch.finally • assert

6.2 Selection Statements

Below are 2 selection statements:

- **if..else**
- **switch case**

Let's see each of these in detail.

6.2.1 if...else

- The **if** statement allows you to execute a block of code if a certain condition is true.

Syntax:

```
if (condition)
    Action if is true
```

Syntax:

```
if (condition) {
    Action if is true
}
else {
    Action if is false
}
```

- The argument to if statement should be boolean type only, else there will be compile-time error.
- **Else part and curly braces are optional.**
- Without curly braces only one statement is allowed under if statement, which should **not be declarative statement**.

- Eg 1:

Code:

```
if (true)
    System.out.println("Hello");
```

- Eg 2:

Code:

```
if (true); ✓// Note: Semicolon is also valid statement.
```

- Eg 3:

Code:

```
if (true)
    int x = 10; ✗// Compile-time error for declarative
statement
```

- Eg 4:

Code:

```
if (true) {
    int x = 10;
}
```

- Eg 5:

Code:

```
int x = 0;
if (x) { ✗// Compile-time error as not a boolean
    System.out.println("Hello");
}
else {
    System.out.println("Hi");
}
```

- Eg 6:

Code:

```
int x = 0;
if (x=20) { ✗// Compile-time error as not a boolean
    System.out.println("Hello");
}
else {
    System.out.println("Hi");
}
```

- Eg 7:

Code:

```
int x = 0;
if (x==20) {
    System.out.println("Hello");
}
else {
    System.out.println("Hi"); // output: Hi
}
```

- Eg 8:

Code:

```
boolean b = false;
if (b == false) {
    System.out.println("Hello"); // output: Hello
}
else {
    System.out.println("Hi");
}
```

Dangling else

- There is no dangling else problem in Java.
- Every else is mapped to the nearest if statement.

Code:

```
if (true)
if (true)
    System.out.println("Hello"); // output: Hello
else
    System.out.println("Hi");
```

6.2.2 switch case

- If several options ar available, then it is not recommended to use nested if..else statement, as it reduces readability.
- Solution: switch statement

Syntax:

```
switch(argument) {
    case arg-1:
        action1;
        break;
    case arg-2:
        action2;
        break;
    case n:
        action-n;
        break
    default:
        Default action
}
```

- Curly braces are mandatory.
- **Case and default are optional**, i.e an empty switch statement is a valid Java syntax.

Code:

```
int x = 10;
switch(x) {} ✓
```

- Allowed argument types in switch statement:
  - Upto Java 1.4 version -> **byte, short, char, int**
  - From Java 1.5 version -> byte, short, char, int, **wrapper classes (Byte, Short, Character, Integer), enum**

- From Java 1.7 versionbyte, short, char, int, wrapper classes (Byte, Short, Character, Integer), enum, **string**

- Inside switch, every statement should be under some case or default.

Code:

```
int x = 10;
switch(x){
    System.out.println(); ✗      // Compile-time error!
}
```

- Case argument should be compile-time constant (i.e constant expression).

Code:

```
int x=10;
int y=20;
switch(x) {
    case 10:
        System.out.println(10);
        break;
    case y; ✗      // Compile-time error!
        System.out.println();
        break;
}
```

Note:

If y is declared as "final", then there will be no compile-time error.

- Switch arguments and case label can be expressions. But, case label should be constant expression.

Code:

```
int x = 10;
switch(x+1) {
    case 10:
        System.out.println(10);
        break;
    case 10+20+30:
        System.out.println(60);
        break;
}
```

- **Case label should be in range of switch arg type**, else it will result in compile-time error.

Eg 1:

Code:

```
byte b = 10;
switch(b) {
    case 10:
        System.out.println(10);
        break;
    case 100:
        System.out.println(100);
        break;
    case 1000: ✗      // Compille-time error!
        System.out.println(1000);
        break;
}
```

Eg 2:

Code:

```
byte b = 10;
switch(b+1) { // Implicit type-caste to integer
    case 10:
        System.out.println(10);
        break;
    case 100:
        System.out.println(100);
        break;
    case 1000:
        System.out.println(1000);
        break;
}
```

- Duplicate case labels are not allowed.

Code:

```
int x = 10;
switch(x) {
    case 12:
        System.out.println(12);
        break;
    case 97:
        System.out.println(97);
        break;
    case 'a': ✗     // Duplicate labels error
        System.out.println(1000);
        break;
}
```

Summary for case-label argument

- It should be constant expression.
- The value should be in the range of switch argument type.
- Duplicate case label are not allowed

Fall-through inside switch

- Within the switch, if any case is matched, from that case onwards all statements will be executed until break or end of the switch.
- This is called fall-through inside switch.
- The main advantage of fall inside the switch is, we can define common action for multiple cases (code-reuseablilty).

Syntax:

```
switch(argument)  {
      case  arg-1:
      case  arg-2:
      case  arg-3:
            action;
            break;
      case  arg-4:
      case  arg-5:
      case  arg-6:
            action;
            break;
}
```

Eg:

Code:

```
int x = 0; switch(x) {
    case 0:
        System.out.println(0);
    case 1:
        System.out.println(1);
        break;
    case 2:
        System.out.println(2);
    default:
        System.out.print("default");
}
```

Output:

```
0
1
```

- **Default case:**
  - Within the switch, **you can take default case at most once**.
  - Default case will be executed if and only if, there is no case matched.
  - Within the switch, you can write default case anywhere but it is recommended to write as last case.

    Eg:

    Code:

    ```
    int x = 3; switch(x) {
        default:
            System.out.println("default")
        case 0:
            System.out.println(0)
        case 1:
    ```

```
            System.out.println(1)
        case 2:
            System.out.println(2)
}
```

Output:

```
default
0
```

6.3 Iteration

Iterative statements allow you to repeat a block of code multiple times. Below are the important iterative statements in Java:

1. while
2. do-while
3. for
4. for-each

6.3.1 while()

- When number of iterations is not known in advance, you should use while loop.

Syntax:

```
while(condition) {
    Action
}
```

- The condition should be of boolean type.

Note:

In Java, "1" is not true or false.

Code:

```
while(1) { ✗
    System.out.println("Hello");
}
```

- Curly braces are optional and without curly you can take only one statement under while, and this statement **should not be declarative**.

- Below are some valid and invalid examples of while:

Code:
```
while(true) ✓
    System.out.println("Hello");
```

Code:
```
while(true); ✓
```

Code:
```
while(true)
    int x = 10; ✗
```

Code:
```
while(true) {
    int int x = 10; ✓
}
```

- Unreachable statement in while loop also results in compile-time error. Below are some examples showing unreachable statement:

Code:
```
while(true) {
    System.out.println("Hello");
}
System.out.println("Hi"); ✗// Compile-time error
```

Code:
```
while(false) {
    System.out.println("Hello");
}
System.out.println("Hi"); ✗// Compile-time error
```

Code:

```
int  a=10,  b=20;
while(a<b)  {  ✓
      System.out.println("Hello");
}
System.out.println("Hi");
```

6.3.2 do-while()

- The partition table in MBR can hold only **4 entries**.
- Hence, there can be only **four partitions in hard disks**.
- These 4 partitions are called primary partitions.
- OS can be installed on the primary partition.

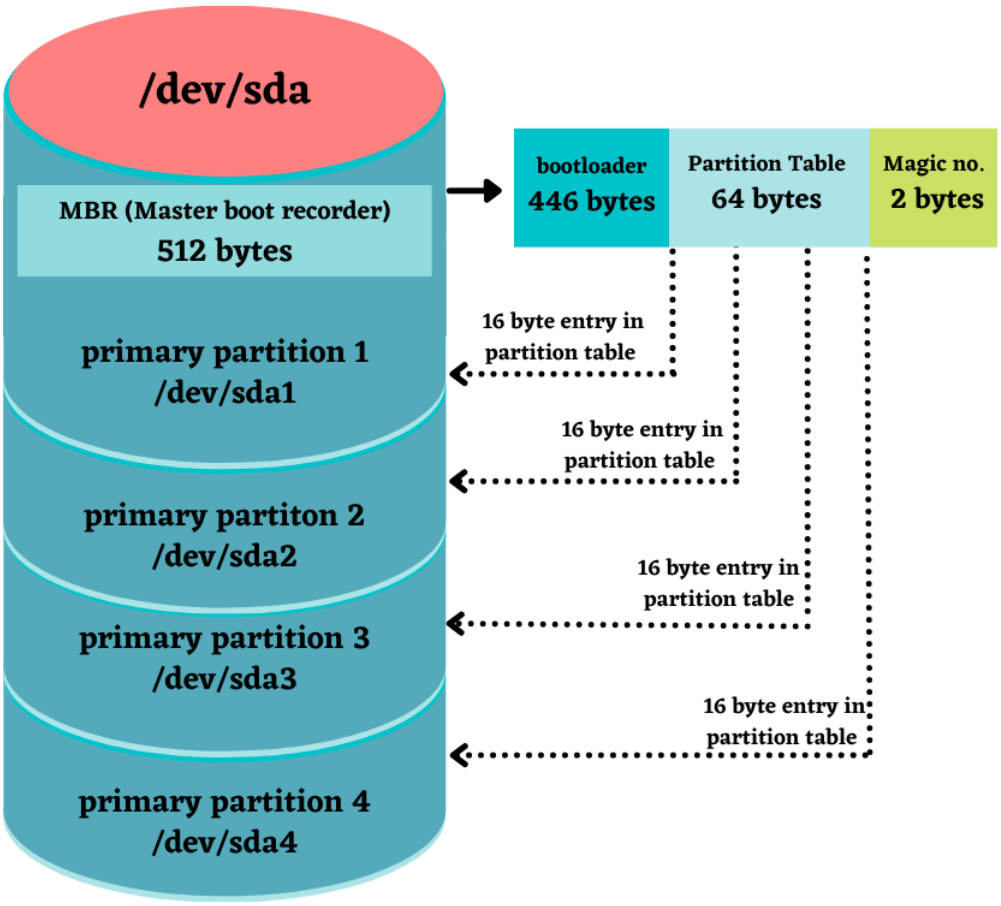


Figure 6.1: Primary partitions

Command to create primary partition

fdisk: Used to change the partition table.

Syntax: fdisk device_name

Options for **fdisk** command:

- **p**: Print the partition table
- **n**: Create a new partition
- **w**: Write the new partition table and exit

Eg:

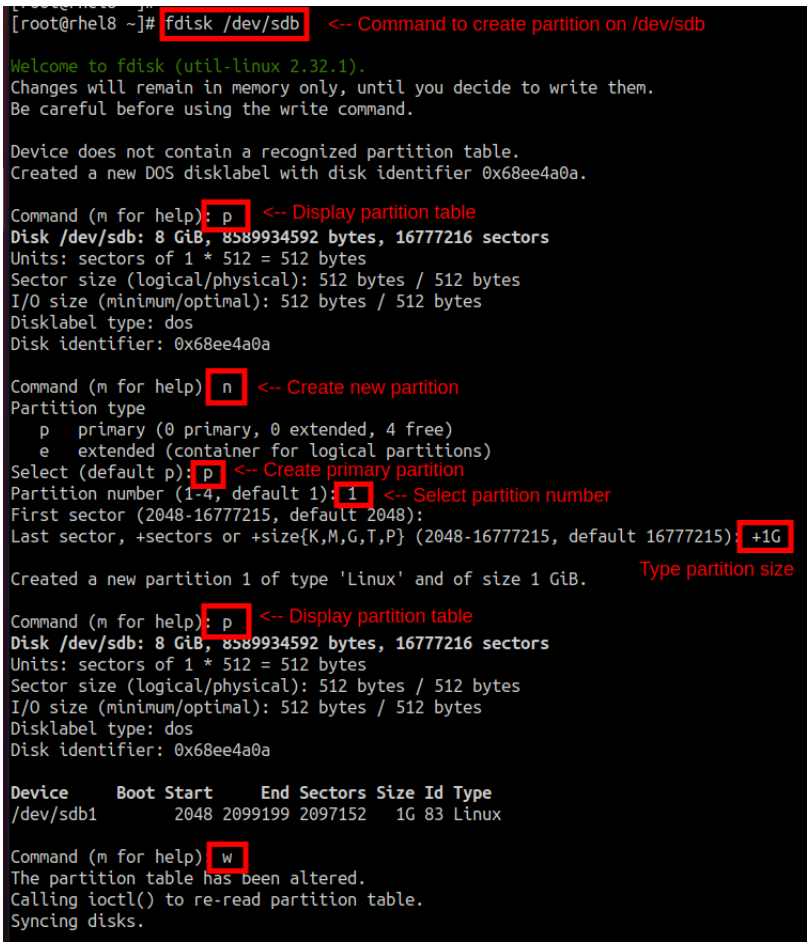


Figure 6.2: Creating primary partitions

6.3.3 for()

- Equality operators can be used for every primitive type including boolean.

Code:

```
System.out.println(10==20); // output: fasle
System.out.println('a' == 'b'); // output: false
System.out.println('a' == 97.0); // output: true
System.out.println(false == false); // output: true
```

- Equality operators can be applied for object types also. For object references, "r1" & "r2", "r1==r2" returns true, if both reference pointing to the same object (reference comparison or address comparison)

Code:

```
Thread t1 = new Thread();
Thread t2 = new Thread();
Thread t3 = t1;
System.out.println(t1 == t2); // output: false
System.out.println(t1 == t3); // output: true
```

There should be some relation between argument types(either child to parent or parent to child or same type). Otherwise, it will result in compile-time error.

Code:

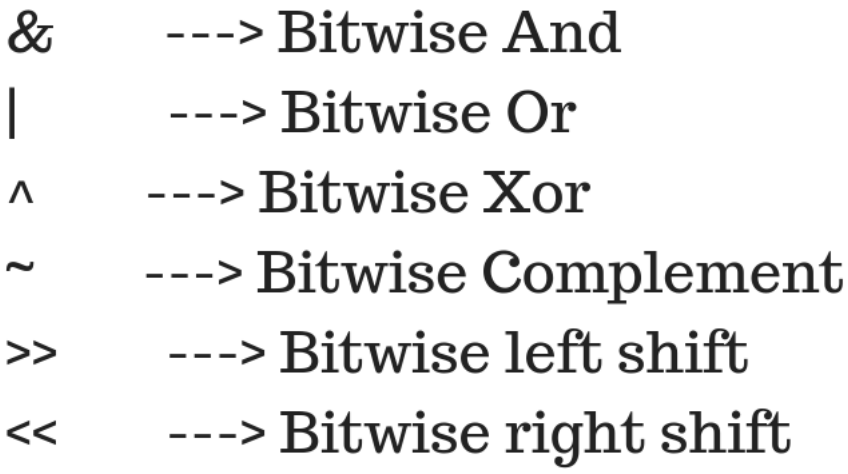
```
Thread t1 = new Thread();
Object o = new Object();
String s = new String("lava");
System.out.println(t1==o) ; // output: false
System.out.println(o==s); // output: false
//System.out.println(s==t1); ✗
```

For any object reference "r": "r==null" ← is always False

Code:

```
String s1 = new String("lava");
System.out.println(s1 == null) ; // output: false

String s2 = new String();
System.out.println(s2==null); // output: false

String s3 = null;
System.out.println(s3==null); // output: true
```

6.3.4 for-each loop



Bitwise & - If both bits are 1, then only 1 otherwise 0

Sample Code	Output	Explaination

int a=4;
int b=5;
System.out.println(a & b)
	4	100
		101
		——
		100

Bitwise | - If atleast one bit is 1, then only 1 otherwise 0

Sample Code	Output	Explaination

int a=4;
int b=5;
System.out.println(a | b)
	5	100
		101
		——
		101

Bitwise ^ - Also called x-or. If both bits are different, then 1, otherwise 0

Sample Code	Output	Explaination

int a=4;
int b=5;
System.out.println(a ^ b)
	1	100
		101
		——
		001

Bitwise ~ - Bitwise complement operator, 1 becomes 0 and 0 becomes 1

Sample Code	Output	Explaination
int a=4; System.out.println(~ a)	-5	32-bit os represents number with total 32 bits as 000...000100 ~ 000...000100 = 111...111011 Left most bit is sign bit where, 1 is negative and 0 is positive Since left most bit is now 1, number is negative Negative number is represented as 1's complement + 1 ie. 100..000100 + 1 = 100...000101

Bitwise » - Bitwise right shift.

Remove "x" bit from right side and add "x" 0 to left side.

Sample Code	Output	Explaination

int a=10;
int x=2;
System.out.println(a » x)
	2	000...1010 » 2
		000...0010

Bitwise « - Bitwise left shift.

Remove "x" bit from left side and add "x" 0 to right side.

Sample Code	Output	Explaination

int a=10;
int x=2;
System.out.println(a « x)
	40	000...1010 « 2
		000...101000

Note:

- &, | , ^ are applicable for both boolean and integral type.
- » , « are applicable for integral type only.
- ~ applicable for only integral type but not for boolean type.

6.4 Transfer Statements

In this section, you are going to learn:

1. **Arithematic Operator**
2. **String Concatenation Operator**
3. **Increment/Decrement Operator**
4. **Relational Operator**
5. **Equality Operator**
6. **Bitwise Operator**
7. **Boolean complement Operator**
8. **Short circuit Operator**
9. **Type cast Operator**
10. **Assignment Operator**
11. **Conditional Operator**
12. **new Operator**
13. **[] Operator**

14.

6.4.1 break

Operator	Example

Addition(+)	`int a = 5;`
	`int b = 10;`
	`int c = a + b; // c will be 15`

Subtraction(-)	`int a = 10;`
	`int b = 5;`
	`int c = a - b; // c will be 5`

Multiplication(*)	`int a = 2;`
	`int b = 3;`
	`int c = a * b; // c will be 6`

Modulus (%)	`int a = 10; int b = 3; int c = a % b; // c will be 1`

Division(/)	int a = 10;
	int b = 3;
	int c = a / b; // c will be 3 (the remainder is discarded)
	
	double d = 10.0;
	double e = 3.0;
	double f = d / e; // f will be 3.33333

Important Points:

- **Implicit type casting:** If we apply any arithmetic operator between 2 variables "a" and "b", the result type is always:

  maximum(int, type of a, type of b)

  Using above formula,

  - Byte + byte = int
  - Byte + short = int
  - Short + short = int
  - Byte + long = long
  - Long + double = double
  - Float + long = float
  - Char + char = int
  - Char + double = double

Eg:

Code:

```
System.out.println('a'+'b'); // output: 195
System.out.println('a'+3.29); // output: 100.29
```

- **Infinity**:
  - In integral arithmetic **(byte short int long), infinity cannot be represented** and JVM will return runtime error.

    Code:

    ```
    System.out.println(10/0); ✗
    ```

  - But in **floating point arithmetic (float, double), infinity can be represented**. For this, Float and Double classes contains below 2 constants:
    - POSITIVE_INFINITY;
    - NEGATIVE_INFINITY;

    Code:

    ```
    System.out.println(10/0.0); // output: Infinity
    System.out.println(-10/0.0); // output: -Infinity
    ```

- **NaN (not a number):**
  - In **integer arithmetic (byte, short, int, long) , undefined results cannot be represented** and JVM will return runtime error.

    Code:

    ```
    System.out.println(0/0); ✗
    ```

  - But, in floating point arithmetic **(float,double), undefined results can be represented as NaN constant**.

Code:

```
System.out.println(0.0/0);  // output:  NaN
System.out.println(-0/0.0);  // output:  NaN
```

6.4.2 continue

Operator	Example

Addition(+)	`int a = 5;`
	`int b = 10;`
	`int c = a + b; // c will be 15`

Subtraction(-)	`int a = 10;`
	`int b = 5;`
	`int c = a - b; // c will be 5`

Multiplication(*)	`int a = 2;`
	`int b = 3;`
	`int c = a * b; // c will be 6`

Modulus (%)	`int a = 10; int b = 3; int c = a % b; // c will be 1`

Division(/)	int a = 10;
	int b = 3;
	int c = a / b; // c will be 3 (the remainder is discarded)
	
	double d = 10.0;
	double e = 3.0;
	double f = d / e; // f will be 3.33333

Important Points:

- **Implicit type casting:** If we apply any arithmetic operator between 2 variables "a" and "b", the result type is always:

  maximum(int, type of a, type of b)

  Using above formula,

  - Byte + byte = int
  - Byte + short = int
  - Short + short = int
  - Byte + long = long
  - Long + double = double
  - Float + long = float
  - Char + char = int
  - Char + double = double

Eg:

Code:

```
System.out.println('a'+'b');  // output:  195
System.out.println('a'+3.29);  // output:  100.29
```

- **Infinity**:
  - In integral arithmetic **(byte short int long), infinity cannot be represented** and JVM will return runtime error.

    Code:

    ```
    System.out.println(10/0);  ✗
    ```

  - But in **floating point arithmetic (float, double), infinity can be represented**. For this, Float and Double classes contains below 2 constants:
    - POSITIVE_INFINITY;
    - NEGATIVE_INFINITY;

    Code:

    ```
    System.out.println(10/0.0);  // output:  Infinity
    System.out.println(-10/0.0);  // output:  -Infinity
    ```

- **NaN (not a number):**
  - In **integer arithmetic (byte, short, int, long) , undefined results cannot be represented** and JVM will return runtime error.

    Code:

    ```
    System.out.println(0/0);  ✗
    ```

  - But, in floating point arithmetic **(float,double), undefined results can be represented as NaN constant**.

Code:

```
System.out.println(0.0/0);  // output:  NaN
System.out.println(-0/0.0);  // output:  NaN
```

6.4.3 return

Operator	Example

Addition(+)	`int a = 5;`
	`int b = 10;`
	`int c = a + b; // c will be 15`

Subtraction(-)	`int a = 10;`
	`int b = 5;`
	`int c = a - b; // c will be 5`

Multiplication(*)	`int a = 2;`
	`int b = 3;`
	`int c = a * b; // c will be 6`

Modulus (%)	`int a = 10; int b = 3; int c = a % b; // c will be 1`

Division(/)	int a = 10;
	int b = 3;
	int c = a / b; // c will be 3 (the remainder is discarded)
	
	double d = 10.0;
	double e = 3.0;
	double f = d / e; // f will be 3.33333

Important Points:

- **Implicit type casting:** If we apply any arithmetic operator between 2 variables "a" and "b", the result type is always:

  maximum(int, type of a, type of b)

  Using above formula,

  - Byte + byte = int
  - Byte + short = int
  - Short + short = int
  - Byte + long = long
  - Long + double = double
  - Float + long = float
  - Char + char = int
  - Char + double = double

Eg:

Code:

```
System.out.println('a'+'b');  // output:  195
System.out.println('a'+3.29);  // output:  100.29
```

- **Infinity**:
  - In integral arithmetic **(byte short int long), infinity cannot be represented** and JVM will return runtime error.

    Code:

    ```
    System.out.println(10/0);  ✗
    ```

  - But in **floating point arithmetic (float, double), infinity can be represented**. For this, Float and Double classes contains below 2 constants:
    - POSITIVE_INFINITY;
    - NEGATIVE_INFINITY;

    Code:

    ```
    System.out.println(10/0.0);  // output:  Infinity
    System.out.println(-10/0.0);  // output:  -Infinity
    ```

- **NaN (not a number):**
  - In **integer arithmetic (byte, short, int, long) , undefined results cannot be represented** and JVM will return runtime error.

    Code:

    ```
    System.out.println(0/0);  ✗
    ```

  - But, in floating point arithmetic **(float,double), undefined results can be represented as NaN constant**.

Code:

```
System.out.println(0.0/0);  // output:  NaN
System.out.println(-0/0.0);  // output:  NaN
```

6.4.4 try..catch..finally

Operator	Example

Addition(+)	`int a = 5;`
	`int b = 10;`
	`int c = a + b; // c will be 15`

Subtraction(-)	`int a = 10;`
	`int b = 5;`
	`int c = a - b; // c will be 5`

Multiplication(*)	`int a = 2;`
	`int b = 3;`
	`int c = a * b; // c will be 6`

Modulus (%)	`int a = 10; int b = 3; int c = a % b; // c will be 1`

Division(/)	int a = 10;
	int b = 3;
	int c = a / b; // c will be 3 (the remainder is discarded)
	
	double d = 10.0;
	double e = 3.0;
	double f = d / e; // f will be 3.33333

Important Points:

- **Implicit type casting:** If we apply any arithmetic operator between 2 variables "a" and "b", the result type is always:

  maximum(int, type of a, type of b)

  Using above formula,

  - Byte + byte = int
  - Byte + short = int
  - Short + short = int
  - Byte + long = long
  - Long + double = double
  - Float + long = float
  - Char + char = int
  - Char + double = double

Eg:

Code:

```
System.out.println('a'+'b');  // output:  195
System.out.println('a'+3.29);  // output:  100.29
```

- **Infinity**:
  - In integral arithmetic **(byte short int long), infinity cannot be represented** and JVM will return runtime error.

    Code:

    ```
    System.out.println(10/0);  ✗
    ```

  - But in **floating point arithmetic (float, double), infinity can be represented**. For this, Float and Double classes contains below 2 constants:
    - POSITIVE_INFINITY;
    - NEGATIVE_INFINITY;

    Code:

    ```
    System.out.println(10/0.0);  // output:  Infinity
    System.out.println(-10/0.0);  // output:  -Infinity
    ```

- **NaN (not a number):**
  - In **integer arithmetic (byte, short, int, long) , undefined results cannot be represented** and JVM will return runtime error.

    Code:

    ```
    System.out.println(0/0);  ✗
    ```

  - But, in floating point arithmetic **(float,double), undefined results can be represented as NaN constant**.

Code:

```
System.out.println(0.0/0);  // output:  NaN
System.out.println(-0/0.0);  // output:  NaN
```

6.4.5 assert

Operator	Example

Addition(+)	`int a = 5;`
	`int b = 10;`
	`int c = a + b; // c will be 15`

Subtraction(-)	`int a = 10;`
	`int b = 5;`
	`int c = a - b; // c will be 5`

Multiplication(*)	`int a = 2;`
	`int b = 3;`
	`int c = a * b; // c will be 6`

Modulus (%)	`int a = 10; int b = 3; int c = a % b; // c will be 1`

Division(/)	int a = 10;
	int b = 3;
	int c = a / b; // c will be 3 (the remainder is discarded)
	
	double d = 10.0;
	double e = 3.0;
	double f = d / e; // f will be 3.33333

Important Points:

- **Implicit type casting:** If we apply any arithmetic operator between 2 variables "a" and "b", the result type is always:

  maximum(int, type of a, type of b)

  Using above formula,

  - Byte + byte = int
  - Byte + short = int
  - Short + short = int
  - Byte + long = long
  - Long + double = double
  - Float + long = float
  - Char + char = int
  - Char + double = double

Eg:

Code:

```
System.out.println('a'+'b');  // output:  195
System.out.println('a'+3.29);  // output:  100.29
```

- **Infinity**:
  - In integral arithmetic **(byte short int long), infinity cannot be represented** and JVM will return runtime error.

    Code:

    ```
    System.out.println(10/0);  ✗
    ```

  - But in **floating point arithmetic (float, double), infinity can be represented**. For this, Float and Double classes contains below 2 constants:
    - POSITIVE_INFINITY;
    - NEGATIVE_INFINITY;

    Code:

    ```
    System.out.println(10/0.0);  // output:  Infinity
    System.out.println(-10/0.0);  // output:  -Infinity
    ```

- **NaN (not a number):**
  - In **integer arithmetic (byte, short, int, long) , undefined results cannot be represented** and JVM will return runtime error.

    Code:

    ```
    System.out.println(0/0);  ✗
    ```

  - But, in floating point arithmetic **(float,double), undefined results can be represented as NaN constant**.

Code:

```
System.out.println(0.0/0);  // output:  NaN
System.out.println(-0/0.0);  // output:  NaN
```

Don't focus on the pain -> 

Focus on progress -> 

- Dwayne Johnson

7. Pandas

7.1 Getting started with Pandas

In this section, you are going to learn text processing commands like:

- **find & grep**
- **head & tail**
- **more & wc**
- **sort, cut & uniq**

There will be a **small excerise** on these topics to check your knowledge.

So let's get started.... 

7.1.1 Pandas Introduction

- Pandas is a Python library used to analyze data.
- It has functions for analyzing, cleaning, exploring, and manipulating data.

What Can Pandas Do?

Pandas gives you answers about the data. Like:

- Is there a correlation between two or more columns?
- What is average value?
- Max value?
- Min value?
- Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called cleaning the data.

Pandas Installation:

- Install pandas package using below command:

Syntax:

```
pip install pandas
```

- Pandas is usually imported under the pd alias.

Code:

```
import pandas as pd
```

7.1.2 Pandas Series

- A Pandas series is like a column in a table.
- It is a one-dimensional array holding data of any type.
- Eg:

Code:

```
import pandas as pd
data = ['apple','mango','chikoo']
var = pd.Series(data)
print(var)
```

Output:

```
0        apple
1        mango
2        chikoo
dtype: object
```

Labels

- If nothing else is specified, the values are labeled with their index number.
- First value has index 0, second value has index 1 etc.

Code:

```
import pandas as pd
data = ['apple','mango','chikoo']
var = pd.Series(data)
print(var[0])
print(var[1])
print(var[2])
```

Output:

```
apple
mango
chikoo
```

- **Create your own labels**

Code:

```
import pandas as pd
a = [1, 7, 2]
myvar = pd.Series(a, index = ["x", "y", "z"])
print(myvar)
print(myvar['x'])
print(myvar['y'])
```

Output:

```
x    1
y    7
z    2
dtype: int64
1
7
```

- **Key/Value Objects as Series:**

Code:

```
import pandas as pd
a = 'name':'Ravi','age':56,'country':'India'
myvar = pd.Series(a)
print(myvar)
```

Output:

```
name                Ravi
age                 56
country             India
dtype: object
```

7.1.3 Pandas DataFrames

- Data sets in Pandas are usually multi-dimensional tables, called **DataFrames**.
- **Series is like a column**, a DataFrame is the whole table.
- Eg:

Code:

```
import pandas as pd
a = {
    'Skill': ['Python','Ruby','Perl'],
    'Candidates': ['Ravi','Ram','Raman']
}
myvar = pd.DataFrame(a)
print(myvar)
```

Output:

	Skill	Candidates
0	Python	Ravi
1	Ruby	Ram
2	Perl	Raman

Locate Row

- Pandas use the **"loc"** attribute to return one or more specified row(s).
- To locate single row:

Code:

```
print(myvar.loc[0])
```

- To locate multiple row:

Code:

```
print(myvar.loc[[0,1]])
```

Example:

Code:

```
import pandas as pd
a = {
    'Skill': ['Python','Ruby','Perl'],
    'Candidates': ['Ravi','Ram','Raman']
}
myvar = pd.DataFrame(a)
print(myvar.loc[0])
print(myvar.loc[1])
```

Output:

```
Skill          Python
Candidates        Ravi
Name: 0, dtype: object
Skill          Ruby
Candidates        Ram
Name: 1, dtype: object
```

- **Named Indexes**

Code:

```
import pandas as pd
a = {
'Skill': ['Python','Ruby','Perl'],
'Candidates': ['Ravi','Ram','Raman']
}
myvar = pd.DataFrame(a,index=['a','b','c'])
print(myvar)
print(myvar.loc['a'])
```

Output:

```
        Skill  Candidates
a      Python        Ravi
b        Ruby         Ram
c        Perl       Raman
Skill      Python
Candidates      Ravi
Name: a, dtype: object
```

7.1.4 Pandas Read CSV

In this chapter, we shall use below dataset to analyse.

Dataset:

```
Series reference,Period,Data value,Suppressed,Group,Stage
BDCQ.SEA1AA,2011.06,80078,,Agriculture,Filled jobs
BDCQ.SEA1AA,2011.09,78324,,Agriculture,Filled jobs
BDCQ.SEA1AA,2011.12,85850,,Agriculture,Not Filled
BDCQ.SEA1AA,2012.03,90743,,Agriculture,Filled jobs
BDCQ.SEA1AA,2012.06,81780,,Agriculture,Filled jobs
BDCQ.SEA1AA,2012.09,79261,,Agriculture,Filled jobs
BDCQ.SEA1AA,2012.12,87793,,Agriculture,Filled jobs
BDCQ.SEA1AA,2013.03,91571,,Agriculture,Filled jobs
BDCQ.SEA1AA,2013.06,81687,,Agriculture,Filled jobs
BDCQ.SEA1AA,2013.09,81471,,Agriculture,Filled jobs
BDCQ.SEA1AA,2013.12,93950,,Industry,Filled jobs
BDCQ.SEA1AA,2014.03,97208,,Agriculture,Filled jobs
BDCQ.SEA1AA,2014.06,85879,,Industry,Not Filled
BDCQ.SEA1AA,2014.09,84447,,Industry,Filled jobs
BDCQ.SEA1AA,2014.12,95075,,Industry,Filled jobs
BDCQ.SEA1AA,2015.03,98202,,Fishing,Filled jobs
BDCQ.SEA1AA,2015.06,87987,,Fishing,Filled jobs
BDCQ.SEA1AA,2015.09,84529,,Fishing,Filled jobs
BDCQ.SEA1AA,2015.12,96848,,Fishing,Filled jobs
BDCQ.SEA1AA,2016.03,99291,,Fishing,Filled jobs
BDCQ.SEA1AA,2016.06,88716,,Industry,Not Filled
BDCQ.SEA1AA,2016.09,85933,,Industry,Not Filled
BDCQ.SEA1AA,2016.12,96540,,Industry,Not Filled
BDCQ.SEA1AA,2017.03,98994,,Fishing,Not Filled
```

- If your data sets are stored in a file, Pandas can load them into a DataFrame.

- CSV files contains plain text and is a well know format that can be read by everyone including Pandas.
- Eg:

Code:

```
import pandas as pd
df = pd.read_csv('data.csv')
print(df)
```

Note:

- By default, if dataset is too large, DataFrame will display first 5 and last 5 lines.
- Use **to_string()** to print the entire DataFrame.

Viewing the Data

- The **head(n)** method returns the headers and a specified number of rows specified using "n", starting from the top.

Code:

```
import pandas as pd
df = pd.read_csv('data.csv')
print(df.head(3))
```

Output:

```
  Series reference    Period  Data value  Suppressed         Group         Stage
0      BDCQ.SEA1AA  2011.06        80078         NaN  Agriculture  Filled jobs
1      BDCQ.SEA1AA  2011.09        78324         NaN  Agriculture  Filled jobs
2      BDCQ.SEA1AA  2011.12        85850         NaN  Agriculture   Not Filled
```

- The **tail(n)** method returns the headers and a specified number of rows, starting from the bottom.

Code:

```
import pandas as pd
df = pd.read_csv('data.csv')
print(df.tail(3))
```

Info About the Data

- The DataFrames object has a method called **info()**, that gives you more information about the data set.

Code:

```
import pandas as pd
df = pd.read_csv('data.csv')
print(df.info())
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31 entries, 0 to 30
Data columns (total 6 columns):
#       Column          Non-Null Count      Dtype
--      ------          --------------      -----
0       Series reference    31 non-null     object
1       Period          31 non-null         float64
2       Data value          31 non-null     int64
3       Suppressed          0 non-null      float64
4       Group           31 non-null         object
5       Stage           31 non-null         object
dtypes: float64(2), int64(1), object(3)
memory usage: 1.6+ KB
None
```

Note:

The info() method also tells us how many Non-Null values there are present in each column, and in our data set it seems like there are 31 Non-Null values in the "Series reference","Period","Data value","Suppressed","Group","Stage" column.

Save data in csv format

- Use **"to_csv()"** to save a dataframe to a ".csv" file.
- Example

Code:

```
import pandas as pd
df = pd.read_csv('data3.csv')
df = df.head(20)
df.to_csv('analysis.csv')
```

7.2 Pandas Deep Dive

In this section, you are going to learn:

1. **What is standard input device & standard output device?**
2. **Output redirection**
3. **Output append operator**
4. **Input redirection**
5. **Error redirection**
6. **Error append redirection**
7. **Output & error redirection**
8. **Output & error append redirection**
9. **Redirection summary**
10. **The pipe operator**

Finally, there will be a **small excerise** on these topics to check your knowledge.

So let's get started....

7.2.1 Cleaning Data

Data cleaning means fixing bad data in your data set. Bad data could be:

- Empty cells
- Data in wrong format
- Wrong data
- Duplicates

Empty Cells

- To delete a specific column that you don't want in the dataframe:

Code:

```
# To delete single column:
df = df.drop('column_name', axis=1)

# To delete multiple column:
df = df.drop(['column_name','column_name'],axis=1)
```

Example:

Code:

```
import pandas as pd
df = pd.read_csv('data3.csv')
print(df.drop('Suppressed',axis=1))
print(df.drop(['Suppressed','Stage'],axis=1))
```

- To delete a specific row that you don't want in the dataframe:

Code:

```
# To delete single row:
df = df.drop('row_index', axis=0)

# To delete multiple row:
df = df.drop(['row_index','row_index'], axis=0)
```

Example:

Code:

```
import pandas as pd
df = pd.read_csv('data3.csv')
print(df.drop(28,axis=0))
print(df.drop([30,29],axis=0))
```

Working with rows having "NaN" cells

- Use **dropna()** function to delete all rows having empty or NaN cells.

Code:

```
import pandas as pd
df = pd.read_csv('data.csv')
print(df.dropna())
```

If you want to change the original DataFrame, use the **"inplace = True"** argument:

Code:

```
import pandas as pd
df = pd.read_csv('data.csv')
df.dropna(inplace=True)
print(df)
```

To replace empty values with some default values, use "**fillna()**" function:

Code:

```
import pandas as pd
df = pd.read_csv('data.csv')
df.fillna(130,inplace=True)
print(df)
```

To replace empty values with some default values but only for a specific column, use "**fillna()**" function:

Code:

```
import pandas as pd
df = pd.read_csv('data.csv')
df["Data value"].fillna(130,inplace=True)
print(df)
```

7.2.2 Pandas Plotting

- Pandas is a Python library used to analyze data.
- It has functions for analyzing, cleaning, exploring, and manipulating data.

What Can Pandas Do?

Pandas gives you answers about the data. Like:

- Is there a correlation between two or more columns?
- What is average value?
- Max value?
- Min value?
- Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called cleaning the data.

Pandas Installation:

- Install pandas package using below command:

Syntax:

```
pip install pandas
```

- Pandas is usually imported under the pd alias.

Code:

```
import pandas as pd
```

Dreams don't work,
unless you do!

8. Appendix

8.1 An A-Z Index of the Linux command line

A

&	Start a new process in the background
alias	Create an alias
apropos	Search Help manual pages (man -k)
apt	Search for and install software packages (Debian/Ubuntu)
apt-get	Search for and install software packages (Debian/Ubuntu)
aptitude	Search for and install software packages (Debian/Ubuntu)
aspell	Spell Checker
at	Schedule a command to run once at a particular time
awk	Find and Replace text, database sort/validate/index

B

basename	Strip directory and suffix from filenames
base32	Base32 encode/decode data and print to standard output
base64	Base64 encode/decode data and print to standard output
bash	GNU Bourne-Again SHell
bc	Arbitrary precision calculator language
bg	Send to background
bind	Set or display readline key and function bindings
break	Exit from a loop
builtin	Run a shell builtin
bzip2	Compress or decompress named file(s)

C

cal	Display a calendar
case	Conditionally perform a command
cat	Concatenate and print (display) the content of files
cd	Change Directory
cfdisk	Partition table manipulator for Linux
chattr	Change file attributes on a Linux file system
chgrp	Change group ownership
chmod	Change access permissions
chown	Change file owner and group
chpasswd	Update passwords in batch mode
chroot	Run a command with a different root directory
chkconfig	System services (runlevel)
cksum	Print CRC checksum and byte counts
clear	Clear terminal screen
cmp	Compare two files
comm	Compare two sorted files line by line
continue	Resume the next iteration of a loop
cp	Copy one or more files to another location
cpio	Copy files to and from archives
cron	Daemon to execute scheduled commands
crontab	Schedule a command to run at a later time
csplit	Split a file into context-determined pieces
curl	Transfer data from or to a server
cut	Divide a file into several parts

D

date	Display or change the date & time
dc	Desk Calculator
dd	Data Duplicator - convert and copy a file, write disk headers, boot records
ddrescue	Data recovery tool
declare	Declare variables and give them attributes
df	Display free disk space
diff	Display the differences between two files
diff3	Show differences among three files
dig	DNS lookup
dir	Briefly list directory contents
dircolors	Colour setup for 'ls'
dirname	Convert a full pathname to just a path
dirs	Display list of remembered directories
dmesg	Print kernel & driver messages
dpkg	Package manager (Debian/Ubuntu).
du	Estimate file space usage

E

echo	Display message on screen
egrep	Search file(s) for lines that match an extended expression
eject	Eject removable media
enable	Enable and disable builtin shell commands
env	Environment variables
ethtool	Ethernet card settings
eval	Evaluate several commands/arguments
exec	Execute a command
exit	Exit the shell
export	Set an environment variable
expr	Evaluate expressions

F

false	Do nothing, unsuccessfully
fdisk	Partition table manipulator for Linux
fg	Send job to foreground
fgrep	Search file(s) for lines that match a fixed string
file	Determine file type
find	Search for files that meet a desired criteria
for	Expand words, and execute commands
format	Format disks or tapes
free	Display memory usage
fsck	File system consistency check and repair
ftp	File Transfer Protocol
function	Define Function Macros
fuser	Identify/kill the process that is accessing a file

G

gawk	Find and Replace text within file(s)
getfacl	Get file access control lists
grep	Search file(s) for lines that match a given pattern
groupadd	Add a user security group
groupdel	Delete a group
groupmod	Modify a group
groups	Print group names a user is in
gzip	Compress or decompress named file(s)

H

hash	Remember the full pathname of a name argument
head	Output the first part of file(s)
help	Display help for a built-in command
history	Command History
hostname	Print or set system name
htop	Interactive process viewer

I

id	Print user and group id's
if	Conditionally perform a command
ifconfig	Configure a network interface
ifdown	Stop a network interface
ifup	Start a network interface up
iostat	Report CPU and i/o statistics
ip	Routing, devices and tunnels

J

jobs	List active jobs

K

kill	Kill a process by specifying its PID
killall	Kill processes by name

L

less	Display output one screen at a time
let	Perform arithmetic on shell variables
link	Create a link to a file
ln	Create a symbolic link to a file
local	Create a function variable
locate	Find files
logname	Print current login name
logout	Exit a login shell
lsattr	List file attributes on a Linux second extended file system
lsblk	List block devices
ls	List information about file(s)
lsof	List open files
lspci	List all PCI devices

M

make	Recompile a group of programs
man	Help manual
mapfile	Read lines from standard input into an indexed array variable
mkdir	Create new folder(s)
mkfifo	Make FIFOs (named pipes)
mkfile	Make a file
mknod	Make block or character special files
mktemp	Make a temporary file
more	Display output one screen at a time
most	Browse or page through a text file
mount	Mount a file system
mv	Move or rename files or directories

N

nc	Necat, read and write data across networks
netstat	Networking connections/stats
nice	Set the priority of a command or job
nslookup	Query Internet name servers interactively

P

passwd	Modify a user password
ping	Test a network connection
pgrep	List processes by name
pkill	Kill processes by name
printenv	Print environment variables
printf	Format and print data
ps	Process status
pwd	Print Working Directory

R

ram	ram disk device
rar	Archive files with compression
rcp	Copy files between two machines
read	Read a line from standard input
readonly	Mark variables/functions as readonly
reboot	Reboot the system
rename	Rename files
renice	Alter priority of running processes
return	Exit a shell function
rm	Remove files
rmdir	Remove folder(s)
rsync	Remote file copy (Synchronize file trees)

S

scp	Secure copy (remote file copy)
sed	Stream Editor
seq	Print numeric sequences
set	Manipulate shell variables and functions
setfacl	Set file access control lists
sftp	Secure File Transfer Program
shutdown	Shutdown or restart linux
sleep	Delay for a specified time
sort	Sort text files
source	Run commands from a file '.'
split	Split a file into fixed-size pieces
ss	Socket Statistics
ssh	Secure Shell client (remote login program)
stat	Display file or file system status
strace	Trace system calls and signals
su	Substitute user identity
sudo	Execute a command as another user
sum	Print a checksum for a file
sync	Synchronize data on disk with memory
systemctl	Command to manage systemd

T

tail	Output the last part of a file
tar	Store, list or extract files in an archive
tee	Redirect output to multiple files
test	Evaluate a conditional expression
time	Measure Program running time
tmux	Terminal multiplexer
touch	Change file timestamps
top	List processes running on the system
traceroute	Trace Route to Host
tr	Translate, squeeze, and/or delete characters
true	Do nothing, successfully
tty	Print filename of terminal on stdin
type	Describe a command

U

ulimit	Limit user resources
umask	Users file creation mask
umount	Unmount a device
unalias	Remove an alias
uname	Print system information
uniq	Uniquify files
unrar	Extract files from a rar archive
unset	Remove variable or function names
uptime	Show uptime
useradd	Create new user account
userdel	Delete a user account
usermod	Modify user account
users	List users currently logged in

V

vi/vim	Text Editor
vmstat	Report virtual memory statistics

W

w	Show who is logged on and what they are doing
wait	Wait for a process to complete
watch	Execute/display a program periodically
wc	Print byte, word, and line counts
whereis	Search the user's $path, man pages and source files for a program
which	Search the user's $path for a program file
while	Execute commands
who	Print all usernames currently logged in
whoami	Print the current user id and name ('id -un')
wget	Retrieve web pages or files via HTTP, HTTPS or FTP

X

xargs	Execute utility, passing constructed argument list(s)
.	Run a command script in the current shell
!!	Run the last command again
#	Comment / Remark

8.2 Technical books from Lavatech Technology

- Learn Linux with Lavatech Technology (System Administration) - Part 1
- Learn Linux with Lavatech Technology (System Engineer) - Part 2
- Learn Python with Lavatech Technology (Core and Advance)
- Learn Ansible with Lavatech Technology
- Learn Docker with Lavatech Technology
- Learn Kubernetes with Lavatech Technology
- Learn complete DevOps with Lavatech Technology
- Learn Jenkins (CI/CD) with Lavatech Technology
- Learn Git/GitHub with Lavatech Technology
- Learn Terraform with Lavatech Technology
- Learn AWS with Lavatech Technology
- Learn Azure with Lavatech Technology

- One
- two
- three
- Four

---

1. Apple
2. Mango
3. Chikoo
4. Pineapple

---

1. Ram
    - **Skill**: Python
    - **Age**: 34
    - **City**: Pune
2. Sham
    - Skill: C++
    - Age: 36
    - City: Mumbai
3. Kavi
    - Skill: C
    - Age: 35
    - City: Bangalore

- Apple
    1. Price: 200
    2. Color: Red
- Mango
    1. Price: 800
    2. Color: Yellow

Datatype	Description
Integer	Contains numbers
FLoat	Contains decimal point
Double	Contains big numbers
String	Contains alphabets
List	Contains list
Tuple	Contains tuple

Datatype	Description	Content
Integer	Contains numbers	1
FLoat	Contains decimal point	2
Double	Contains big numbers	Aple
String	Contains alphabets	Mango
List	Contains list	Chikoo
Tuple	Contains tuple	Banana