# Basics of java

* Variables and data types

Variables: **Variable in Java** is a data container that stores the data values during Java program execution

1. Local variables
2. Instance variables
3. Static variables

Example: class Demo {

static int a = 1; //static variable

int data = 99; //instance variable

void method() {

int b = 90; //local variable

}

}

Data types: **Data Types in Java** are defined as specifiers that allocate different sizes and types of values that can be stored in the variable or an identifier

* 1. Primitive datatypes: which include integer, character, boolean, and float
  2. Non primitive datatypes: which include classes, arrays and interfaces.

|  |  |  |
| --- | --- | --- |
| Java Data Types | | |
| **Data Type** | **Default Value** | **Default size** |
| byte | 0 | 1 byte |
| short | 0 | 2 bytes |
| int | 0 | 4 bytes |
| long | 0L | 8 bytes |
| float | 0.0f | 4 bytes |
| double | 0.0d | 8 bytes |
| boolean | false | 1 bit |
| char | ‘\u0000’ | 2 bytes |

* Arrays
  1. Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.
  2. To declare an array, define the variable type with **square brackets:**

Syntax: String[] cars;

* Control flow statements
  + If-else:

Example:

int time = 20;

if (time < 18) {

System.out.println("Good day.");

} else {

System.out.println("Good evening.");

}

* + Switch case:

Syntax:

switch(*expression*) {

case x:

*// code block*

break;

case y:

*// code block*

break;

default:

*// code block*

}

# Methods

* A **method** is a block of code which only runs when it is called.
* You can pass data, known as parameters, into a method.
* Methods are used to perform certain actions, and they are also known as **functions**.
* Why use methods? To reuse code: define the code once, and use it many times.

Method creation inside the main:

public class Main {

static void myMethod() {

// code to be executed

}

}

# Object orientation

* Class and object
  + Java is an object-oriented programming language.
  + Everything in Java is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an object. The car has **attributes**, such as weight and color, and **methods**, such as drive and brake.

Example:

To create a class, use the keyword class:

Create a class named "Main" with a variable x:

public class Main {

int x = 5;

}

Example:

In Java, an object is created from a class. We have already created the class named Main, so now we can use this to create objects.

To create an object of Main, specify the class name, followed by the object name, and use the keyword new:

Create an object called "myObj" and print the value of x:

public class Main {

int x = 5;

public static void main(String[] args) {

Main myObj = new Main();

System.out.println(myObj.x);

}

}

* Inheritance

In Java, it is possible to inherit attributes and methods from one class to another. We group the "inheritance concept" into two categories:

* **subclass** (child) - the class that inherits from another class
* **superclass** (parent) - the class being inherited from

To inherit from a class, use the extends keyword.

In the example below, the Car class (subclass) inherits the attributes and methods from the Vehicle class (superclass):

* Interface

Another way to achieve abstraction in Java, is with interfaces.

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

* Abstract classes

Data abstraction is the process of hiding certain details and showing only essential information to the user.

Abstraction can be achieved with either abstract classes or interfaces (which you will learn more about in the next chapter).

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

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

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

An abstract class can have both abstract and regular methods:

# Loops

* While

The while loop loops through a block of code as long as a specified condition is true:

Syntax:

while (condition) {

// code block to be executed

}

* For

When you know exactly how many times you want to loop through a block of code, use the for loop instead of a while loop:

* Nested loops and debugging

Nested loop is said to be the loop inside another loop.debugging is said to be that verify weather the code is run properly or not ,and we can keep break points and identify the execution flow.

## Running programs with command line

C:\Users\Your Name>javac Main.java

C:\Users\Your Name>java Main

Output: Hello World

## Creating jar files using command line

javac HelloWorld.java

### **Creating an Executable JAR File**

Main-Class: HelloWorld

Next, we can create a JAR file by running the following jar command.

jar -cfm HelloWorld.jar ManifestFile.txt HelloWorld.class

The **-c flag** is used to create an archive file. The **-f flag** is used to specify the file name. And the **-m flag** will include the content of the manifest file.

### **Creating a Non-Executable JAR File**

To create a non-executable JAR file, we will exclude the **-m flag**. We don't need to pass the name of the manifest file to our command.

jar -cf HelloWorld.jar HelloWorld.class

## **Running JAR Files**

### **Running the Executable JAR File**

We can use the following java command with the **-jar** option to run the executable JAR file.

java -jar HelloWorld.jar

The output of the command is shown below.

Hello world!

### **Running the Non-Executable JAR File**

Instead of using the **-jar option**, we will use the **-cp option** to run a **non-executable JAR** file. We need to specify the JAR file name and the main class name in the command.

java -cp HelloWorld.jar HelloWorld

The output of this command is shown below.

Hello world!

# File processing and exception handling

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | **class** Exception{  **public** **static** **void** main(String args[]){  **try**{  //code that may raise exception  }  **catch**(Exception e){  // rest of the program    }   }  } |

Basic example of execption:

* Try

The try block contains a set of statements where an exception can occur. It is always followed by a catch block, which handles the exception that occurs in the associated try block. A try block must be followed by catch blocks or finally block or both.

Syntax:

try{

//code that may throw exception

}catch(Exception\_class\_Name ref){}

* Catch

A catch block is where you handle the exceptions. This block must follow the try block and a single try block can have several catch blocks associated with it.

* Finally

A finally block contains all the crucial statements that must be executed whether an exception occurs or not. The statements present in this block will always execute, regardless an exception occurs in the try block or not such as closing a connection, stream etc.

Example:

**class** SampleFinallyBlock{

**public** **static** **void** main(String args[]){

**try**{

**int** data=55/5;

     System.out.println(data);

    }

**catch**(NullPointerException e)

       {System.out.println(e);}

**finally** {System.out.println("finally block is executed");}

    System.out.println("remaining code");

  }

}

# Collections frame work

* + - Array list

Example:

import java.util.\*;

class TestJavaCollection1{

public static void main(String args[]){

ArrayList<String> list=new ArrayList<String>();//Creating arraylist

list.add("Ravi");//Adding object in arraylist

list.add("Vijay");

list.add("Ravi");

list.add("Ajay");

//Traversing list through Iterator

Iterator itr=list.iterator();

while(itr.hasNext()){

System.out.println(itr.next());

}

}

}

* + - Linked list

Example:

import java.util.\*;

public class TestJavaCollection2{

public static void main(String args[]){

LinkedList<String> al=new LinkedList<String>();

al.add("Ravi");

al.add("Vijay");

al.add("Ravi");

al.add("Ajay");

Iterator<String> itr=al.iterator();

while(itr.hasNext()){

System.out.println(itr.next());

}

}

}

* + - Vector

Example:

import java.util.\*;

public class TestJavaCollection3{

public static void main(String args[]){

Vector<String> v=new Vector<String>();

v.add("Ayush");

v.add("Amit");

v.add("Ashish");

v.add("Garima");

Iterator<String> itr=v.iterator();

while(itr.hasNext()){

System.out.println(itr.next());

}

}

}

* + - Hash set

Example:

import java.util.\*;

public class TestJavaCollection7{

public static void main(String args[]){

//Creating HashSet and adding elements

HashSet<String> set=new HashSet<String>();

set.add("Ravi");

set.add("Vijay");

set.add("Ravi");

set.add("Ajay");

//Traversing elements

Iterator<String> itr=set.iterator();

while(itr.hasNext()){

System.out.println(itr.next());

}

}

}

* + - Linked hash set

Example:

import java.util.\*;

public class TestJavaCollection8{

public static void main(String args[]){

LinkedHashSet<String> set=new LinkedHashSet<String>();

set.add("Ravi");

set.add("Vijay");

set.add("Ravi");

set.add("Ajay");

Iterator<String> itr=set.iterator();

while(itr.hasNext()){

System.out.println(itr.next());

}

}

}

# Java generics

* Generics

Generics means parameterized types. The idea is to allow type (Integer, String, … etc., and user-defined types) to be a parameter to methods, classes, and interfaces. Using Generics, it is possible to create classes that work with different data types.

Type Parameters in Java Generics:

1. T – Type
2. E – Element
3. K – Key
4. N – Number
5. V – Value

BaseType <Type> obj = new BaseType <Type>()

* Generics with wildcard

The question mark (?) is known as the wildcard in generic programming. It represents an unknown type. The wildcard can be used in a variety of situations such as the type of a parameter, field, or local variable; sometimes as a return type.

Types of wildcards in Java:

1. Upper Bounded Wildcards:

public static void add(List<? extends Number> list)

1. Lower Bounded Wildcards:

Syntax: Collectiontype <? super A>

# Concurrency in java

* Threads and runnable interface
* Thread safety with synchronization
* Thread safety with collections

# JDBC

* Preparing database and basic SQL commands
* Use jdbc to send sql statements after connection
* Curd operations using jdbc

## Lambda expressions