Java:

\* It is an Object Oriented Programming language developed by

James Gosling.

Object Object Programming: Representation of realtime entities in

the form of objects , providing the communication between them to

provide a solution for a realtime problem.

Java can be divided into 3 editions.

1. JSE - Java Standard Edition - Desktop Applications

2. JEE - Java Enterpise Edition - Web Applications / Enterprise Apps

3. JME - Java Mobile / Micro Edition - Mobile Apps.

Environment Set up:

1. install jdk, jre

<https://www.oracle.com/in/java/technologies/downloads/#jdk17-windows>

JDK: Java Development Kit - contains the required tools and jar files

for application development.

jar - java archive file - it contains the required packages (classes and interfaces).

JRE: Java Runtime Environment -provides the Environment to run java applications.

JVM: Java Virtual Machine - responsible for running the java application in

JRE.

First Application using Java:

class FirstApp {

public static void main(String[] args){

System.out.println("Hello World!!");

}

}

\* The filename and classname need not to be the same (however, it is recommended to

have the same name).

06032023:

Object Oriented Programming:

OOPs is a paradigm where all the real time objects are visualized using

java objects , their relationship to solve a realtime problem.

Real time Object: Any physical entity that has state and action can be

considered as realtime object.

We represent realtime objects by creating java objects using a template

called class.

Building blocks of Java:

Variables:

A variable is an identifier which is used to identify some data that

can be used in the application.

syntax:

datatype var\_name = value;

In Java, we have 3 types of variables.

1. instance / non static variables :

A variable that is created without a static keyword and is accessed using

an object is known as non static variable.

since these variables are bound to object,hence we can call them as

instance variables.

code\_snippet:

public class InstanceVariablesDemo {

int age=21;

String name= "kiran";

String email = "kiran@gmail.com";

public static void main(String[] args) {

InstanceVariablesDemo obj1 = new InstanceVariablesDemo();

System.out.println(obj1.age);

System.out.println(obj1.name);

System.out.println(obj1.email);

obj1.age=23;

obj1.name= "kiran kumar";

InstanceVariablesDemo obj2 = new InstanceVariablesDemo();

System.out.println(obj2.age); // 21

System.out.println(obj2.name); // kiran

}

}

Problem Statement:

ABC bank would like to provide online banking facility to its customers and

new customers can also create a new account and use the services.

1. withdraw

2. deposit

Object: Customer

properties: name, accNo,accountBalance,address....

action: withdraw, deposit

Solution:

class Customer{

String name;

String accNo;

long accBalance;

String address;

public void withdraw(long amount) {

this.accBalance = this.accBalance - amount;

}

public void deposit(long amount) {

this.accBalance = this.accBalance + amount;

}

}

public class BankingApp {

public static void main(String[] args) {

Customer customer1= new Customer();

customer1.name = "kiran";

customer1.accNo = "12356";

customer1.accBalance = 9000;

customer1.address ="Hyderabad";

Customer customer2= new Customer();

customer2.name = "rajesh";

customer2.accNo = "878788";

customer2.accBalance = 10000;

customer2.address ="Hyderabad";

System.out.println(customer1.accBalance);

customer1.deposit(9000);

System.out.println(customer1.accBalance);

customer2.withdraw(2000);

System.out.println(customer2.accBalance);

}

}

2. static variables

3. local variables

07032023:

Static Variables:

A variable that is associated with a class is known as static variable / class

variable.

We use static keyword before the variable creation to create a static

variable.

static Variables maintains only one copy of the memory which means modification

of the static variable in a class can effect the original value.

The static variables will be created outside of a method.

public class StaticVariablesDemo {

static int a=10;

static String b="hello";

public static void main(String[] args) {

System.out.println(a);

System.out.println(StaticVariablesDemo.a); // recommended

System.out.println(StaticVariablesDemo.b); // recommended

StaticVariablesDemo obj = new StaticVariablesDemo();

System.out.println(obj.a);

System.out.println(obj.b);

obj.a = 200;

System.out.println(StaticVariablesDemo.a); //

StaticVariablesDemo.b = "World";

System.out.println(obj.b); //

}

}

example:

public class LikesDemo {

static int noOfLikes = 0;

public static void main(String[] args) {

LikesDemo user1 = new LikesDemo();

user1.noOfLikes++;

LikesDemo user2 = new LikesDemo();

LikesDemo user3 = new LikesDemo();

LikesDemo user4 = new LikesDemo();

user2.noOfLikes++;

user3.noOfLikes++;

user4.noOfLikes++;

user4.noOfLikes--;

System.out.println("Likes :"+ noOfLikes);

}

}

differences between static and non static variables:

class Product{

static String pid; // remove static and see the output

static String name;

static long price; // 0

}

public class ProductDemo {

public static void main(String[] args) {

Product p1= new Product();

p1.pid="101x";

p1.price=2000;

p1.name = "Prod1";

System.out.println(p1.pid);

System.out.println(p1.price);

System.out.println(p1.name);

System.out.println("---------------------");

Product p2= new Product();

p2.pid="200";

p2.price=4000;

p2.name = "Prod2";

System.out.println(p2.pid);

System.out.println(p2.price);

System.out.println(p2.name);

System.out.println("---------------------");

System.out.println(p1.pid); //

System.out.println(p1.price);

System.out.println(p1.name);

}

}

Local variables:

If we create any variable inside a method, then it is known as

local variable.

public class LocalVariablesDemo {

public static void m1() {

int a=100; // local variable

System.out.println(a);

}

public static void main(String[] args) {

LocalVariablesDemo.m1();

}

}

Task:

Create a practical example differentiating local variables, static

variables and non static variables.

08032023:

Data Types:

There are 2 types of data types.

1. primitive types:

Java provides 8 primitive data types.

Data Type Default Value Default size

boolean false 1 bit

char '\u0000' 2 byte

byte 0 1 byte - 2power7 - 1 = 128 - 1 = 127

short 0 2 byte - 2power15 - 1

int 0 4 byte - 2power31 - 1

long 0L 8 byte

float 0.0f 4 byte

double 0.0d 8 byte

public class DataTypes {

// Primitive data types

static boolean booleanType;

static char charType;

static byte byteType;

static short shortType;

static int intType;

static long longType;

static float floatType;

static double doubleType;

public static void main(String[] args) {

System.out.println(booleanType);

System.out.println(charType);

System.out.println(byteType);

System.out.println(shortType);

System.out.println(intType);

System.out.println(longType);

System.out.println(floatType);

System.out.println(doubleType);

System.out.println(Byte.MAX\_VALUE);

System.out.println(Byte.MIN\_VALUE);

System.out.println(Short.MAX\_VALUE);

System.out.println(Short.MIN\_VALUE);

System.out.println(Integer.MAX\_VALUE);

System.out.println(Long.MAX\_VALUE);

}

}

2. non primitive / advanced types:

Classes and Interfaces can be considered as non primitive data types.

ex: Customer customer = new Customer();

Customer customer;

From above Customer can be called as Non primitive data type.

// Non Primitive types

static String username;

static DataTypes dataTypes;

Full Example:

import java.util.ArrayList;

public class DataTypes {

// Primitive data types

static boolean booleanType;

static char charType;

static byte byteType;

static short shortType;

static int intType;

static long longType;

static float floatType;

static double doubleType;

// Non Primitive types

static String username;

static DataTypes dataTypes;

public static void main(String[] args) {

System.out.println(booleanType);

System.out.println(charType);

System.out.println(byteType);

System.out.println(shortType);

System.out.println(intType);

System.out.println(longType);

System.out.println(floatType);

System.out.println(doubleType);

System.out.println(Byte.MAX\_VALUE);

System.out.println(Byte.MIN\_VALUE);

System.out.println(Short.MAX\_VALUE);

System.out.println(Short.MIN\_VALUE);

System.out.println(Integer.MAX\_VALUE);

System.out.println(Long.MAX\_VALUE);

System.out.println(username);

System.out.println(dataTypes);

}

}

Methods:

A method is a block of code that takes an input (optional), process it

and provides some result.

syntax:

datatype nameOfMethod(parameters){

// code

}

nameOfMethod(arguments);

int square(int number){

return number \* number;

}

square(10);

There are 2 types of methods in Java.

1. static

2. non static

public class MethodsDemo {

static byte a=100;

short b=200;

void nonStaticMethod() {

System.out.println("Non Static Method");

}

void secondNonStaticMethod() {

nonStaticMethod();

staticMethod();

System.out.println("second non static method");

System.out.println(MethodsDemo.a);

System.out.println(b);

}

static void staticMethod() {

System.out.println("Static method executed");

}

static void secondStaticMethod() {

System.out.println(MethodsDemo.a);

System.out.println(new MethodsDemo().b);

}

public static void main(String[] args) {

// MethodsDemo.staticMethod();

MethodsDemo demo = new MethodsDemo();

// demo.nonStaticMethod();

demo.secondNonStaticMethod();

MethodsDemo.secondStaticMethod();

}

}

another example:

public class MethodsDemo {

static byte a=100;

short b=200;

void nonStaticMethod() {

System.out.println("Non Static Method");

}

void secondNonStaticMethod() {

nonStaticMethod();

staticMethod();

System.out.println("second non static method");

System.out.println(MethodsDemo.a);

System.out.println(a);

System.out.println(b);

}

static void staticMethod() {

System.out.println("Static method executed");

}

static void secondStaticMethod() {

System.out.println(MethodsDemo.a);

System.out.println(new MethodsDemo().b);

}

public static void main(String[] args) {

// MethodsDemo.staticMethod();

MethodsDemo demo = new MethodsDemo();

// demo.nonStaticMethod();

demo.secondNonStaticMethod();

MethodsDemo.secondStaticMethod();

}

}

09032023:

Control Statements:

sequential : if, if-else and switch

iterative : while, do-while and for

jump: continue, return, break

sequential : if, if-else and switch

if-else:

syntax:

if(test-condition){

// code

}

else{

// code

}

import java.util.Scanner;

public class Sequential {

public static void main(String[] args) {

Scanner sc= new Scanner(System.in);

System.out.println("Enter username");

String username = sc.next();

System.out.println("Enter password");

String password = sc.next();

System.out.println(username + "\t" + password);

if(username.contentEquals("digitallync") && password.contentEquals("digitallync@123")) {

System.out.println("login success");

}

else {

System.out.println("login failure");

}

sc.close();

}

}

switch:

if we have more than two conditions, then we can use switch.

syntax:

switch(test-condition){

case case\_value1 : // code

case case\_value2 : //code

case case\_value3: //code

.

.

.

default: // code

}

String choice= "java";

switch(choice) {

case "java": System.out.println("You chose java"); break;

case "dotnet": System.out.println("You chose dotnet"); break;

case "react": System.out.println("You chose react"); break;

case "angular": System.out.println("You chose angular"); break;

default : System.out.println("wrong choice");

}

We can have numbers, string, characters as choices in a switch case.

Iterative Statements:

while:

syntax:

while(test-condition){

// code

}

we use while to iterate a list of items (array).

int index = 0;

while(index < 10) {

System.out.println(index);

index++;

}

String[] users= {"user1","user2","user3","user4","user5"};

int index= 0;

while(index < users.length) {

System.out.println(users[index]);

index++;

}

class Product{

int productId;

String name;

long price;

String seller;

}

public class IterativeStatements {

public static void main(String[] args) {

Product product1 = new Product();

product1.productId = 101; product1.name="product 1"; product1.price=9000; product1.seller="seller 1";

Product product2 = new Product();

product2.productId = 102; product2.name="product 2"; product2.price=2000; product2.seller="seller 2";

Product product3 = new Product();

product3.productId = 103; product3.name="product 3"; product3.price=3000; product3.seller="seller 3";

Product product4 = new Product();

product4.productId = 104; product4.name="product 4"; product4.price=6000; product4.seller="seller 2";

Product[] products = {product1, product2,product3, product4};

int index = 0;

while(index < products.length) {

Product product = products[index];

System.out.println(product.productId + "\t"+product.name + "\t"+ product.price+"\t"+ product.seller);

index++;

}

}

}

do-while:

if we want some block to be executed once irrespective of test condition,

then we can choose do-while.

do{

// code

}while(test-condition)

listof items (100000): API call (2 minutes) - 28 seconds

split of data into equal parts : 1000 items ( 5 seconds )

int allItemsSize = items.length; // 10000

do {

// API call - returns first 1000 items along with the items size (10000)

// 2000

int currentCount = 10000;

}

while(currentCount < allItemsSize)

14032023:

For loop:

syntax:

for(initialization;test-condition; inc / dec){

}

for loop is an enhancement of while loop.

public class ForLoopDemo {

public static void main(String[] args) {

String users[]= {"user1","user2","user3","user4","user5"};

for(int index = 0; index < users.length; index++) {

System.out.println(users[index]);

}

}

}

Jump Statements:

break: to terminate a block of code, we can use break statement.

continue: If we want to skip an iteration, then we can use continue.

public class ForLoopDemo {

public static void main(String[] args) {

String users[]= {"user1","user2","user3","user4","user5"};

for(int index = 0; index < users.length; index++) {

if(users[index].contentEquals("user3")) {

continue;

}

System.out.println(users[index]);

}

}

}

Difference between break and continue:

break is used to terminate a block of code

continue is used to skip a specific iteration

return:

There are some functions / methods that act as helper methods which always

return result of some operation.

so we use return from that helper method /function.

import java.io.ObjectInputStream.GetField;

import java.util.Scanner;

public class JumpStatements {

// helper methods

static boolean authenticateUser(String username, String password) {

if(username.contentEquals("digitallync") && password.contentEquals("digitallync")) {

return true;

}

else {

return false;

}

}

static int generateOtp(boolean isUserLoggedIn) {

int otp = 0;

if(isUserLoggedIn) {

// code to generate OTP;

otp = 1234;

}

return otp;

}

public static void main(String[] args) {

Scanner sc= new Scanner(System.in);

System.out.println("Enter username");

String username = sc.next();

System.out.println("Enter password");

String password = sc.next();

boolean result=authenticateUser(username, password);

if(result) {

System.out.println("user logged in successfully!!");

}

else {

System.out.println("username/ password incorrect!");

}

System.out.println(generateOtp(true));

}

}

15032023:

Operators:

An Operator is a special symbol that provides some functionality.

example: +

10 + 20 = 30

+ operator performs addition of two numbers.

"hello" + "world" = hello world

+ operator performs concatenation of two strings.

Arithmetic Operators:

+,-,\*,/(quotient),%(reminder)

public class Arithmetic {

public static void main(String[] args) {

String str="100";

int a=10;

System.out.println(str + 100); // 100100

System.out.println(str + 100 + 10);

System.out.println(a + 100 + str); // 110100

}

}

unary operators:

++ (increment), -- (decrement)

if we use ++ before a variable, it is called pre increment

if we use ++ after a variable, it is called post increment

if we use -- before a variable, it is called pre decrement

if we use -- after a variable, it is called post decrement

public class Unary {

public static void main(String[] args) {

int a=100;

System.out.println(a++); // prints 100, a=101

System.out.println(++a); // a=102, prints 102

System.out.println(--a); // a=101, prints 101

System.out.println(a--); //prints 101 and a=100

System.out.println(a); //100

}

}

public class Unary {

public static void main(String[] args) {

int a=100;

++a;

System.out.println(a); // 101

a++;

System.out.println(a); // 102

--a;

System.out.println(a); // 101

a--;

System.out.println(a); // 100

a++;

System.out.println(a); // 101

//logger - log

}

}

Comparision / Relational:

>, <,>=,<=, !=, ==

class Customer{

int id;

String name;

}

public class Comparision {

public static void main(String[] args) {

int a=100;

int b=200;

String c="100";

Comparision c1= new Comparision(); // addr -> x1

Comparision c2 = new Comparision(); // addr -> x2

Comparision c3 = c2;

System.out.println(a==b);

System.out.println(a>b);

System.out.println(a<b);

System.out.println(c1);

System.out.println(c2);

System.out.println(c1 == c2); // false

System.out.println(c2 == c3); //

Customer customer1= new Customer();

customer1.id=101;

Customer customer2= new Customer();

customer2.id=101;

System.out.println(customer1 == customer2);

System.out.println(customer1.id == customer2.id);

System.out.println(customer1.id != customer2.id);

}

}

16032023:

Object Oriented Programming:

If a Programming language is object oriented, then the following features

are applicable.

1. inheritance

2. polymorphism

3. abstraction

4. encapsulation

Constructors:

A Constructor is a special method that constructs an object.

While the object is being constructed, the constructor gets invoked.

without creation of a constructor, we cannot create object.

The constructor name is same as classname.

syntax:

nameoftheclass(){

//code

}

if there is no constructor in the class, then a default constructor is

provided.

class Customer {

int cid;

String name;

long accountBalance;

}

public class ConstructorDemo {

public static void main(String[] args) {

Customer customer = new Customer(); // calling default constructor

System.out.println(customer.cid);

}

}

We can use constructors to initialize the data for the instance variables at

runtime.

class Customer {

int cid;

String name;

long accountBalance;

// parameterized constructor

public Customer(int cid, String name, long accountBalance) {

this.cid = cid;

this.name= name;

this.accountBalance = accountBalance;

}

}

public class ConstructorDemo {

public static void main(String[] args) {

Customer customer = new Customer(101, "kiran",909999l); // calling default constructor

System.out.println(customer.cid + "\t" + customer.name+"\t"+ customer.accountBalance);

}

}

this: is a keyword which refers to currently executing object.

class Customer {

int cid;

String name;

long accountBalance;

// parameterized constructor

public Customer(int id, String name, long accountBalance) {

this.cid = id;

this.name = name;

this.accountBalance = accountBalance;

}

}

public class ConstructorDemo {

public static void main(String[] args) {

Customer customer1 = new Customer(101, "kiran",909999l); // cid, name, accountBalance

Customer customer2 = new Customer(102, "ravi",909l); // cid, name, accountBalance

Customer customer3 = new Customer(103, "anil",11909l); // cid, name, accountBalance

System.out.println(customer1.cid + "\t" + customer1.name+"\t"+ customer1.accountBalance);

System.out.println(customer2.cid + "\t" + customer2.name+"\t"+ customer2.accountBalance);

}

}

Constructor Overloading:

In a class, if we want a constructor to provide multiple functionalities then

we recreate the construcor using constructor Overloading.

In Constructor Overloading, The following rules are applied.

1. there should be a difference in number of arguments.

2. there should be a difference in type of arguments.

class Customer {

int cid;

String name;

long accountBalance;

String idProof;

String incomeProof;

String dobCertificate;

// parameterized constructor

public Customer(int id, String name, long accountBalance,String idProof, String incomeProof,String dobCertificate) {

this.cid = id;

this.name = name;

this.accountBalance = accountBalance;

}

// overloaded constructor

public Customer(int id, String name, long accountBalance,String idProof) {

this.cid = id;

this.name = name;

this.accountBalance = accountBalance;

this.idProof = idProof;

}

}

public class ConstructorDemo {

public static void main(String[] args) {

Customer customer = new Customer(101,"xyz",90900L,"adhaar");

Customer customer2 = new Customer(102, "jsbdc",888l,"adhaar","someincomeproof","dob ceritificate");

}

}

Inheritance:

Inheritance is the process where a class extends another class to reuse

the resources (variables, functions etc).

class Bank{

String bankName = "ICICI";

String branch = "Madhapur";

}

class Customer extends Bank { // Customer -> sub class , Bank -> super class

int cid;

String name;

public String toString() {

// TODO Auto-generated method stub

return this.cid + "\t"+ this.name + "\t"+ this.bankName+"\t"+ this.branch;

}

}

public class InheritanceDemo {

public static void main(String[] args) {

Customer customer = new Customer();

customer.cid=101;

customer.name="kiran";

System.out.println(customer);

Customer customer2 = new Customer();

customer2.cid=102;

customer2.name="rajesh";

System.out.println(customer2);

}

}

17032023:

Types of Inheritance:

We have the following types of Inheritance.

1. Single :

class A{

}

class B extends A{

}

B is sub class and A is super class.

Sub class can access the properties / resources of its own class and super class.

super class can access only its properties / resources which means The super class cannot access sub class

properties.

class Parent{

int a=100;

int b=200;

}

class Child extends Parent {

int c=300;

int d=400;

}

public class SingleInheritanceDemo {

public static void main(String[] args) {

Child child = new Child();

System.out.println(child.c+"\t"+ child.d + "\t"+ child.a +"\t"+ child.b);

Parent parent = new Parent();

System.out.println(parent.a);

}

}

2. Multiple :

class A{

}

class B{

}

class C{

}

class D extends A,B,C{ // Multiple Inheritance

}

Java does not support multiple Inheritance.

class A{

int a=100;

}

class B{

int a=300;

}

class C extends A,B{

syso(a); //

}

From the above, class C cannot decide on which a to be called, hence the ambiquity occurs.

To aviod any kind of ambiquities, Java does not support multiple Inheritance.

3. hierarchical :

hierarchical Inheritance is opposite of multiple Inheritance.

class A{

}

class B extends A{

}

class C extends A{

}

class D extends A{

}

4. multilevel: In multilevel, A sub class can become a super class and go to the levels.

class A{

}

class B extends A{

}

class C extends B{

}

class D extends C{

}

5. hybrid:

Any combination of 2 Inheritances is known as hybrid Inheritance.

In hybrid Inheritance, the multiple Inheritance is not considered.

Constructor cannot be Inherited however, super() keyword always calls

the super class constructor.

class A{

A(){

System.out.println("Constructor A() called");

}

}

class B extends A{

B(){

super();

System.out.println("Costructor B() called");

}

}

public class ConstructorInheritance {

public static void main(String[] args) {

B b= new B();

}

}

class Bank{

String bankName;

String branch;

Bank(String bankName, String branch){

this.bankName = bankName;

this.branch = branch;

}

}

class Customer extends Bank{

String name;

String accId;

public Customer(String name, String accId, String bankName, String branch) {

// TODO Auto-generated constructor stub

super(bankName,branch);

this.name= name;

this.accId = accId;

}

}

public class ConstructorInheritance {

public static void main(String[] args) {

Customer customer = new Customer("kiran","12344","hdfc","hitech city");

System.out.println(customer.bankName + "\t"+ customer.name);

}

}