Course Contents

1. Core Java
2. JDBC
3. Hibernate
4. Spring Core
5. Spring MVC
6. Spring REST
7. Spring Boot
8. Spring Data JPA
9. Spring Microservices

Softwares:

1. JDK 1.8
2. Eclipse IDE for Enterprise Developers
3. Database - Derby/MySQL

Java:

Java is platform independent and Object-Oriented programming language.

Platform Independent:

You can run java program on multiple platforms without altering (without recompiling).

Object Oriented Program:

Object oriented program allows you to create real world entities in the applications and make an object communicate with another object to complete any task.

Ex: In banking applications you can have objects like Customer, Employee, Account, Loan and etc.

Object will have 2 things

1. Properties: What object has like name, dob, phone, …
2. Behaviours: What object does like debit(), withdraw(), getInformations(),…

2 Building blocks of any object-oriented language are:

1. Class: It is a blue print/template for an object
2. Object: It is an instance of a class

HelloApp.java

**package** com.org;

**public** **class** HelloApp {

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

System.***out***.println("Welcome to eclipse");

**int** x = 10;

System.***out***.println("x = "+x); System.***out***.println("x = "+x);

}

}

Fundamentals of Java

* Datatypes
* Operators
* Conditional Statements
* Looping Constructs
* Arrays
* Classes & Objects

Datatypes: It specifies what kind of value a variable can store, there are 8 primitive datatypes in Java

1. byte
2. short
3. int
4. long
5. float
6. double
7. char
8. boolean

byte, short, int & long can store integral values i.e., whole numbers but their size varies

byte - 1 byte: -128 to -1 and 0 to 127  
short - 2 bytes: -32768 to -1 and 0 to 32767  
int - 4 bytes  
long - 8 bytes

int x = 25;  
long y = 35;

float & double can store floating point values float takes 4 bytes & double takes 8 bytes

double a = 20.25;

float b = 20.25f; // since 20.25 is considered as double type you need to explicitly use 20.25f, so that compiler treats the value as float type.

char stores single character its size is 2 bytes

char c = ‘M’;

boolean stores either true/false, its size is 1 byte

boolean b = true;

All the above types belong to primitive datatypes.

Primitive types mean it can store single value/simple values

We have another type in datatypes i.e., derived types which can store objects

Naming Convention

Classnames: Must begin with uppercase & follow camel case

Variables & Methods: Must begin with lowercase & follow camel case

Constructors: Must begin with uppercase & follow camel case

Packages: Must begin with lowercase & use . to separate the package & sub-package

TestDatatypes.java

**package** com.org;

**public** **class** TestDatatypes {

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

**byte** byteValue = 127;

System.***out***.println("byteValue: "+byteValue);

**float** floatValue = 20.5f;

System.***out***.println("floatValue: "+floatValue);

**boolean** booleanValue = **true**;

System.***out***.println("booleanValue: "+booleanValue);

**char** gender = 'M';

System.***out***.println("gender: "+gender);

gender = 'F';

System.***out***.println("gender: "+gender);

String name = "Alexandar";

System.***out***.println("name: "+name);

}

}

Output:

byteValue: 127

floatValue: 20.5

booleanValue: true

gender: M

gender: F

name: Alexandar

Operators: It is used to perform the operations on the variables like arithmetic operations, logical operations

Some of the operations are:

+, -, \*, /, =, <, >, <=, >=, ==, ++, --, &&, ||

TestOperators.java

**package** com.org;

**public** **class** TestOperators {

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

**double** amount = 25000;

**double** balance = 20000;

**if**(amount <= balance) {

System.***out***.println("Amount can be withdrawn");

balance = balance - amount;

} **else** {

System.***out***.println("Amount exceeds the balance");

}

System.***out***.println("Balance: "+balance);

}

}

Output:

Amount exceeds the balance

Balance: 20000.0

Conditional Statements

Some of the conditional statements

1. if
2. if - else
3. if - else if else if … else
4. switch

TestIfElseIf.java

**package** com.org;

**public** **class** TestIfElseIf {

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

**int** marks = 60;

**if**(marks >= 70) {

System.***out***.println("FCD");

} **else** **if** (marks >= 60 && marks < 70) {

System.***out***.println("First class");

} **else** **if** (marks >= 50 && marks < 60) {

System.***out***.println("Second class");

} **else** **if** (marks >= 35 && marks < 50) {

System.***out***.println("Pass");

} **else** {

System.***out***.println("Fail");

}

}

}

Output:

First class

Looping constructs

For loop

While loop

Do while loop

TestLoop.java

**package** com.org;

**public** **class** TestLoop {

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

**for**(**int** i = 1; i <= 10; i++) {

System.***out***.println("i = "+i);

}

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

**int** counter = 10; // 0

**while**(counter > 0) { // 0 > 0

System.***out***.println("Counter: "+counter);

counter--;

}

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

counter = 10;

**do** {

System.***out***.println("Do-while counter: "+counter);

counter--;

} **while**(counter > 0);

}

}

We can also use arrays and iterate the arrays using loops

Array: It is a container to store multiple values in a single variable

int marks1 = 55, marks2 = 75, ….

int[] marks = {55, 75,….}

TestArrays.java

**package** com.org;

**public** **class** TestMarks {

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

**int**[] marks = {75, 80, 70};

// array starts from 0th index, it has length property

System.***out***.println("Size: "+marks.length);

**for**(**int** i = 0; i < marks.length; i++) { // i < 3

System.***out***.println("marks: "+marks[i]); // i = 2, marks[2]

}

}

}

Output:

Size: 3

marks: 75

marks: 80

marks: 70

Classes & Objects

Class is blue print of an object or template of an object, it doesn’t take space on the heap

Object is a real-world entity or instance of a class.

What all the things we can write in a class?

1. Variables - properties
2. Methods - behaviours
3. Constructors - like methods but name will be same as class name which is used to initialize the object.

Employee.java

**package** com.org;

**public** **class** Employee {

**int** id;

String name;

**double** salary;

**void** display() {

System.***out***.println("Id = "+id);

System.***out***.println("Name = "+name);

System.***out***.println("Salary = "+salary);

}

}

Note: By default every class will have the default constructor which doesn’t take any parameter, the default constructor is created by the compiler

Note: Default constructor is created only when explicitly no constructors are present in the class

TestEmployee.java

**package** com.org;

**public** **class** TestEmployee {

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

// creating object of Employee class

// data-type variableName = value;

// int x = 20;

Employee e1 = **new** Employee();

Employee e2 = **new** Employee();

e1.id = 100;

e1.name = "Alex";

e1.salary = 35200.0;

e2.id = 200;

e2.name = "Bruce";

e2.salary = 42000.0;

// invoke display e1.display() & e2.display()

e1.display();

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

e2.display();

}

}

Constructors with parameters

StudentConstructor.java

**package** com.org;

**public** **class** StudentConstructor {

**int** rollNo;

String name;

**char** grade;

StudentConstructor(**int** r, String n, **char** g) {

System.***out***.println("StudentConstructor(int, String, char) called");

rollNo = r;

name = n;

grade = g;

}

**void** display() {

System.***out***.println("RollNo = "+rollNo+", Name = "+name+", Grade = "+grade);

}

}

TestStudent.java

**package** com.org;

**public** **class** TestStudent {

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

StudentConstructor s1 = **new** StudentConstructor(100, "Alex", 'C');

StudentConstructor s2 = **new** StudentConstructor(200, "Bruce", 'A');

s1.display();

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

s2.display();

}

}

Output:

StudentConstructor(int, String, char) called

StudentConstructor(int, String, char) called

RollNo = 100, Name = Alex, Grade = C

--------------------

RollNo = 200, Name = Bruce, Grade = A

Note: In the above example we are avoiding the initialization of properties for every object

Constructor Overloading:

Having more than one constructor’s in the class

Exercise:

1. Create User class with 4 properties
   1. name
   2. age
   3. phone
   4. email
2. Create 4 constructors to initialize the variables
   1. One constructor to initialize name & age
   2. One constructor to initialize name, age, phone
   3. One constructor to initialize name, age, email
   4. One constructor to initialize name, age, phone, email
3. Create a display to show the user properties