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

Types of variables in Java

We have 3 types of variables

1. Instance Variables: Every object will have separate copies of instance variable
2. Static Variables: All the objects share single copy of static variables, you will use class name to access however you can also use object reference to access the static variables
3. Local Variables/Parameter Variables: It is local to the particular scope.

User.java

**package** com.org;

**public** **class** User {

**int** id;

**static** **int** *counter*;

}

TestVariables.java

**package** com.org;

**public** **class** TestVariables {

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

User user1 = **new** User(); // user1[id = 0]

User user2 = **new** User(); // user2[id = 0]

user1.id = 20;

user2.id = 30;

User.*counter* = 1000;

System.***out***.println("user1 id = "+user1.id+", user2 id = "+user2.id);

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

}

}

Output:

user1 id = 20, user2 id = 30

Counter = 1000

Accessing the static variables using classname & object reference

**package** com.org;

**public** **class** TestVariables {

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

User user1 = **new** User(); // user1[id = 0]

User user2 = **new** User(); // user2[id = 0]

user1.id = 20;

user2.id = 30;

User.*counter* = 1000;

user1.*counter* = 2000;

user2.*counter* = 3000;

System.***out***.println("user1 id = "+user1.id+", user2 id = "+user2.id);

System.***out***.println("Counter = "+User.*counter*); // 3000

System.***out***.println("Counter = "+user1.*counter*); // 3000

System.***out***.println("Counter = "+user2.*counter*); // 3000

}

}

Output:

user1 id = 20, user2 id = 30

Counter = 3000

Counter = 3000

Counter = 3000

Final keyword: It can be used for variables, methods & classes

Final keyword used for variables will become constant, it can’t be modified.

final int x = 15;

Here x value will be fixed to 15, it can’t be modified, doing so will give compilation error.

Keyword this: it is used to avoid naming conflicts between the instance variables & local variables when their names are same.

Person.java

**package** com.org;

**public** **class** Person {

String name;

**int** age;

Person(String name, **int** age) {

**this**.name = name;

**this**.age = age;

}

**void** display() {

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

System.***out***.println("Name = "+**this**.name+", Age = "+**this**.age);

}

}

TestPerson.java

**package** com.org;

**public** **class** TestThisKeyword {

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

Person p1 = **new** Person("Alex", 35); // p1[name=Alex & age=35]

Person p2 = **new** Person("Bruce", 40); // p2 [name=Bruce & age=40]

p1.display();

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

p2.display();

}

}

Output:

Name = Alex, Age = 35

Name = Alex, Age = 35

================

Name = Bruce, Age = 40

Name = Bruce, Age = 40

Features of Object Oriented Language

1. Inheritance
2. Encapsulation
3. Polymorphism
4. Abstraction

Inheritance: It is a process of acquiring the properties & behaviours of an object from another object

Ex: Consider the classes like Employee, Student, Customer, having common properties & behaviours

1. Employee -
   1. id, name, salary, phone, email
   2. updateName(), updateSalary(), updatePhone(), updateEmail()
2. Customer -
   1. customerId, name, accountNo, phone, email
   2. updateName(), updatePhone(), updateEmail()
3. Student -
   1. rollNo, name, phone, email, grade
   2. updateName(), updatePhone(), updateEmail(), updateGrade()

If you notice the above classes, you can observe some properties & behaviours are repeating like name, phone, email, updatePhone(), updateName(), updateEmail(), instead of writing them in each class you can create one super class and inherit properties & behaviours of super class/parent class into the sub-class/child class.

Overriding: Method names and signature will be same but implementation will be different.

Advantage of overriding is it can implement the specific feature in the sub-class instead of using the super class feature

Person.java

**package** com.org;

**public** **class** Person {

String name;

**int** age;

Person(String name, **int** age) {

**this**.name = name;

**this**.age = age;

}

**void** display() {

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

System.***out***.println("Name = "+**this**.name+", Age = "+**this**.age);

}

}

Employee.java

**package** com.inheritance;

**public** **class** Employee **extends** Person {

// name, phone, display() are inheritted

**int** id;

**double** salary;

@Override

**void** display(){

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

// it calls the display of super class

**super**.display();

}

}

Student.java

**package** com.inheritance;

**public** **class** Student **extends** Person {

// name, phone, display()

**int** rollNo;

String grade;

@Override

**void** display() {

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

**super**.display();

}

}

TestInheritance.java

**package** com.inheritance;

**public** **class** TestInheritance {

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

Person p = **new** Person();

p.name = "Alex";

p.phone = 998811235;

p.display();

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

Employee e = **new** Employee();

e.id = 100;

e.name = "Bruce";

e.phone = 88998899;

e.salary = 45000;

e.display();

e.updatePhone(88888);

e.display();

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

Student s = **new** Student();

s.rollNo = 200;

s.name = "Charles";

s.phone = 7766688;

s.grade = "A";

s.display();

s.updatePhone(6666);

s.display();

}

}

Output:

---- display of Person -----

Name = Alex, Phone = 998811235

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

Id = 100, Salary = 45000.0

---- display of Person -----

Name = Bruce, Phone = 88998899

Id = 100, Salary = 45000.0

---- display of Person -----

Name = Bruce, Phone = 88888

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

RollNo = 200, Grade = A

---- display of Person -----

Name = Charles, Phone = 7766688

RollNo = 200, Grade = A

---- display of Person -----

Name = Charles, Phone = 6666

Note: every subclass constructor by default calls their parent class default constructor, however you can use super() to call parameterized constructor.

Person.java

**package** com.inheritance;

**public** **class** Person {

String name;

**long** phone;

Person(String name, **long** phone) {

**this**.name = name;

**this**.phone = phone;

System.***out***.println("Person(2) constructor");

}

**void** display() {

System.***out***.println("---- display of Person -----");

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

}

**void** updatePhone(**long** phone) {

**this**.phone = phone;

}

}

Employee.java

**package** com.inheritance;

**public** **class** Employee **extends** Person {

// name, phone, display() are inheritted

**int** id;

**double** salary;

Employee(**int** id, String name, **double** salary, **long** phone) {

**super**(name, phone); // Person(name, phone)

**this**.id = id;

**this**.salary = salary;

System.***out***.println("Employee(4) constructor");

}

@Override

**void** display(){

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

// it calls the display of super class

**super**.display();

}

}

Student.java

**package** com.inheritance;

**public** **class** Student **extends** Person {

// name, phone, display()

**int** rollNo;

String grade;

Student(**int** rollNo, String name, **long** phone, String grade) {

**super**(name, phone);

**this**.rollNo = rollNo;

**this**.grade = grade;

System.***out***.println("Student(4) constructor");

}

@Override

**void** display() {

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

**super**.display();

}

}

TestInheritance.java

**package** com.inheritance;

**public** **class** TestInheritance {

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

Person p = **new** Person("Alex", 99989393);

p.display();

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

Employee e = **new** Employee(100, "Bruce", 235200, 8888899);

e.display();

e.updatePhone(88888);

e.display();

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

Student s = **new** Student(200, "Charles", 9999000, "C");

s.display();

s.updatePhone(6666);

s.display();

}

}

Output:

Person(2) constructor

---- display of Person -----

Name = Alex, Phone = 99989393

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

Person(2) constructor

Employee(4) constructor

Id = 100, Salary = 235200.0

---- display of Person -----

Name = Bruce, Phone = 8888899

Id = 100, Salary = 235200.0

---- display of Person -----

Name = Bruce, Phone = 88888

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

Person(2) constructor

Student(4) constructor

RollNo = 200, Grade = C

---- display of Person -----

Name = Charles, Phone = 9999000

RollNo = 200, Grade = C

---- display of Person -----

Name = Charles, Phone = 6666

Note: super() calls must always be in the first line of the constructor

Note: Object class is the super most class for all the classes, it means if a class doesn’t explicitly extend any class then it will automatically extend Object class

Encapsulation:

Hiding the data, you will wrap the data and code in a single unit and hide the data from the outside world and you can access the data only within the class.

What is the purpose of hiding the data (Variables)?

To control the data so that you can prevent the data to be misused/giving invalid values.

Ex: You want employeeId, customerId to be only seen but prevent from modification

Customer.java

**package** com.encapsulation;

**public** **class** Customer {

**private** **int** customerId;

**private** String name;

**private** **long** phone;

**public** Customer(**int** customerId, String name, **long** phone) {

**super**();

**this**.customerId = customerId;

**this**.name = name;

**this**.phone = phone;

}

// setter method to modify the name

**public** **void** setName(String name) {

**this**.name = name;

}

// setter method to modify the phone

**public** **void** setPhone(**long** phone) {

**this**.phone = phone;

}

// getter method to retrieve the name

**public** String getName() {

**return** name;

}

// getter method to retrieve the id

**public** **int** getCustomerId() {

**return** customerId;

}

// getter method to retrieve the phone

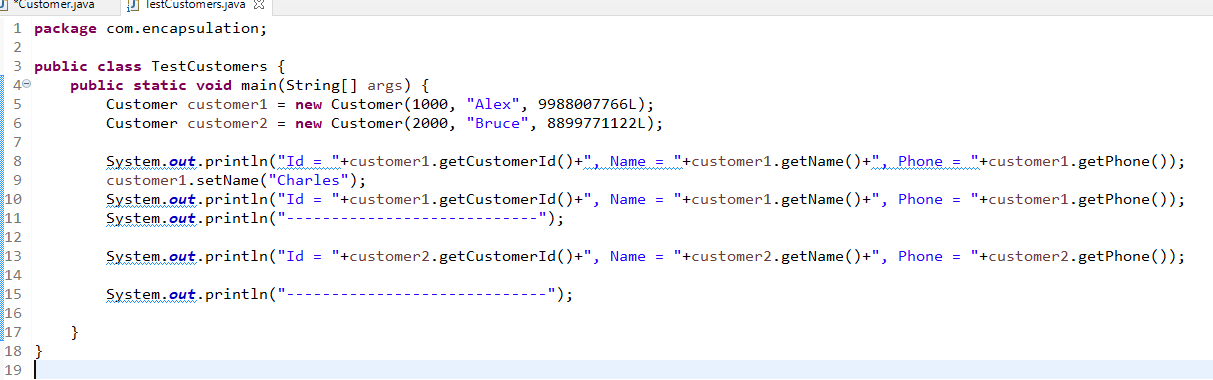
**public** **long** getPhone() {

**return** phone;

}

}

TestCustomers.java



Output:

Id = 1000, Name = Alex, Phone = 9988007766

Id = 1000, Name = Charles, Phone = 9988007766

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

Id = 2000, Name = Bruce, Phone = 8899771122

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

Short cut:

For Constructors: Right Click -> Source -> Generate constructor

For Getters & Setters: Right Click -> Source -> Generate setters & getters

Polymorphism:

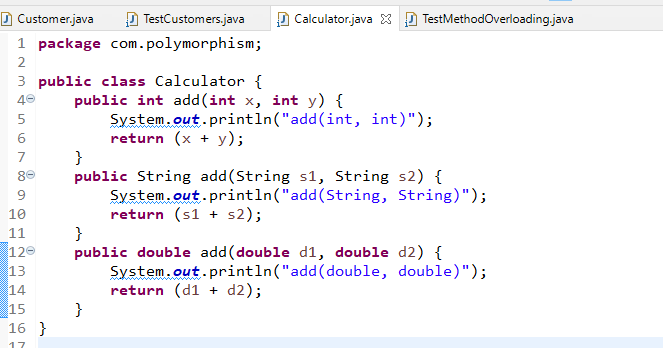
An object with many forms, a single method which can perform multiple task

Ex: TV power button -> on/off

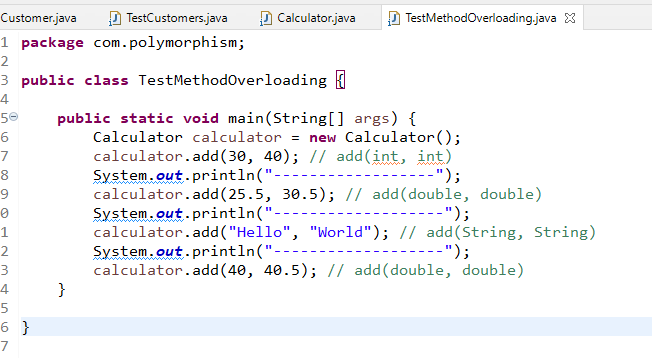
Polymorphism can be achieved in two ways

1. Method overloading (Compile time polymorphism) - at the compile time only you can find which method is called
2. Method overriding (Runtime polymorphism) - only at runtime you can find which method is called

Calcualator.java



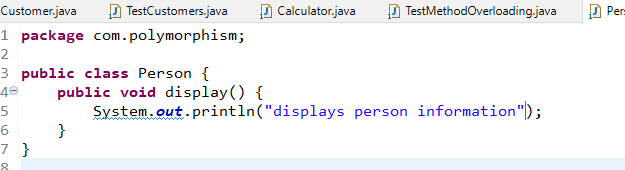
TestMethodOverloading.java



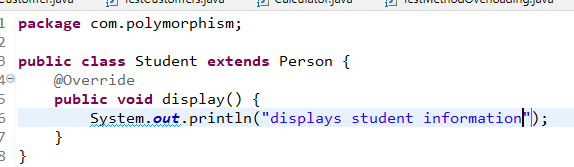
Runtime polymorphism

Here a method calling can be decided only when you know which object is calling it.

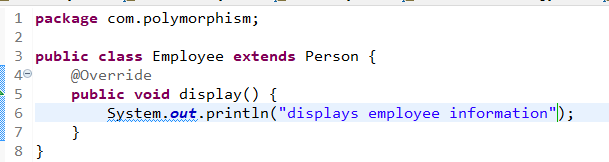
Person.java



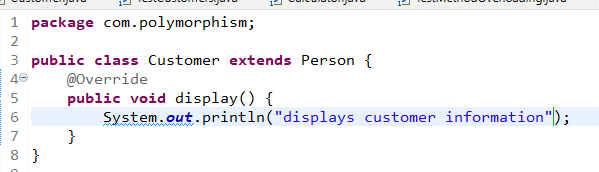
Student.java



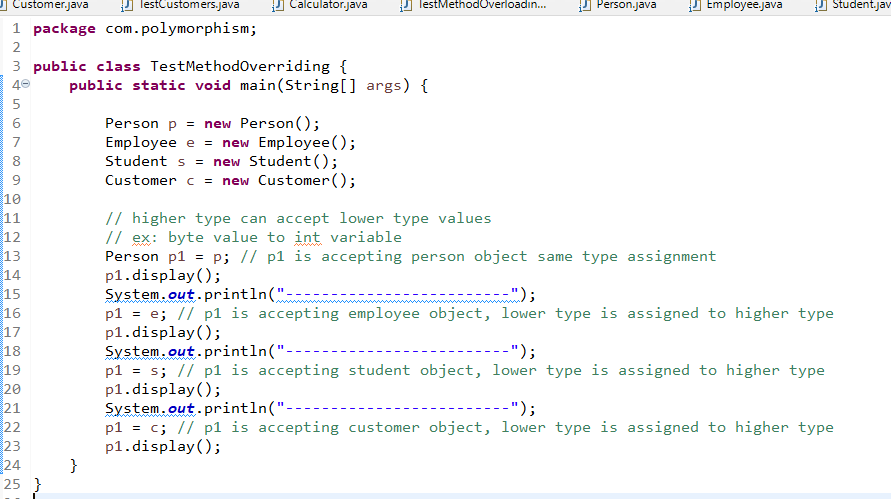
Employee.java



Customer.java



TestMethodOverriding.java



Output:

displays person information

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

displays employee information

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

displays student information

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

displays customer information

Abstraction:

Hiding the complexity and showing the necessary interface/details to the user, here the benefits you get would be the developers can concentrate on their code without understanding other’s code & also if any changes done in the implementation the caller’s code need not be modified.

Abstraction makes developers to call each other code without affecting their code.

Abstraction can be achieved in 2 ways

1. Interfaces - complete abstraction
2. Abstract classes - Partial abstrction

Interface: It will have only abstract methods, i.e., methods without logic & it will also have constants

Note: By default all the members of interfaces are public

Abstract class: It will have both abstract methods & complete methods, abstract class may not have full implementation.

Note: You can’t create object of interface & abstract class

Vehicle.java

**package** com.abstraction;

**public** **interface** Vehicle {

**void** mileage(); // by default methods are abstract

**abstract** **void** wheels(); // abstract is optional

}

TestAbstraction.java

**package** com.abstraction;

**abstract** **class** Bike **implements** Vehicle {

**public** **void** wheels() {

System.***out***.println("Bike has 2 wheels");

}

}

**class** Pulsar **extends** Bike {

**public** **void** mileage() {

System.***out***.println("Pulsar gives 40kmpl");

}

}

**class** RoyalEnfield **extends** Bike {

**public** **void** mileage() {

System.***out***.println("RoyalEnfield gives 30kmpl");

}

}

**public** **class** TestAbstraction {

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

Vehicle v;

Pulsar p = **new** Pulsar();

RoyalEnfield r = **new** RoyalEnfield();

v = p;

v.mileage(); // pulsar mileage

v.wheels(); // pulsar wheels

v = r;

v.mileage(); // royal enfield mileage

v.wheels(); // royal enfield wheels

}

}

Differences between interfaces and abstract class

|  |  |
| --- | --- |
| Interfaces | Abstract class |
| All the methods are by default abstract | All the methods are not abstract by default because you can have abstract & non-abstract methods |
| All the members are public | By default nothing is public |
| Variables are constants by default i.e., static final | Not all variables are static & final by default |
| We can’t write constructors | We can have constructors |
| You can’t create object of an interface | You can’t create object of an abstract class |

The above differences holds good till Java 7, because from java 8 onwards you can have default methods & static methods in interface

Differences between overloading & overriding

|  |  |
| --- | --- |
| Method Overloading | Method Overriding |
| Happens in the same class | Happens in the subclass |
| Method name will be same but signature will be different like return type, number parameters, parameter types | Method name and method signature will be same, but logics/implementation will be different |

Predefined packages & classes

Packages means folders it will have sub-packages & classes, with packages you can arrange classes in a better way & also packages will give you better encapsulation i.e., accessibility or visibility.

Mainly there 4 access specifiers

1. Private
2. Package or Default: No keyword
3. Protected
4. Public

|  |  |  |  |
| --- | --- | --- | --- |
| Private | Default | Protected | Public |
| Within the class | Within the package but outside the package not visible | Within the package but outside the package visible only to the subclass | Visible to all |

Predefined packages & classes in Java

1. java.lang: The classes present in java.lang can be used without importing
2. java.util
3. java.time
4. java.sql

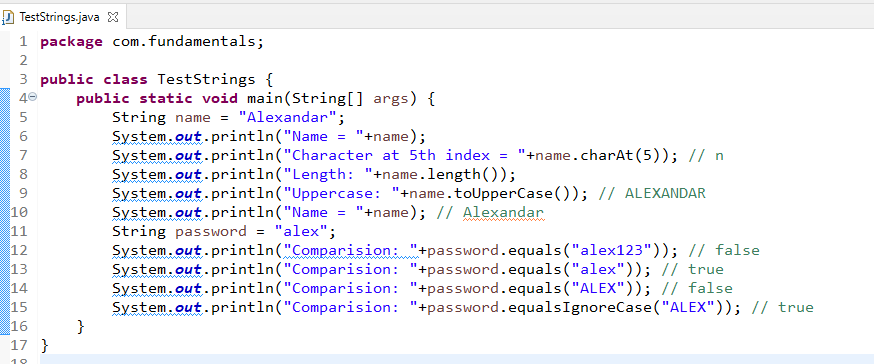
java.lang

We have classes in java.lang like

1. String
2. Object
3. Integer
4. Float
5. Double
6. System
7. Exception
8. ArithmeticException
9. NumberFormatException

String class  
It is used to work on strings like you can concatenate, compare, change string to uppercase, lowercase, you can find length of the string and so on

TestString.java



Object class: It is the root class, every class is a subclass of Object, many classes have overridden some of the methods of Object class.

Most of the classes has overridden some of the common methods present in object class

1. toString(): It is used to return the string representation of the object, by default it returns memory address
2. equals(): It is used compare two objects: ex: In String class it is overridden to compare strings
3. hashCode(): It is used to return the unique number of an object

Employee.java

**package** com.fundamentals;

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

**private** **int** id;

**private** String name;

**private** **double** salary;

**public** Employee(**int** id, String name, **double** salary) {

**super**();

**this**.id = id;

**this**.name = name;

**this**.salary = salary;

}

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **double** getSalary() {

**return** salary;

}

**public** **void** setSalary(**double** salary) {

**this**.salary = salary;

}

@Override

**public** String toString() {

**return** "Employee [id=" + id + ", name=" + name + ", salary=" + salary + "]";

}

}

TestEmployee.java

**package** com.fundamentals;

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

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

String name = "Alexandar"; // string object

System.***out***.println(name); // toString() method of String

Employee e1 = **new** Employee(100, "Raj", 56000);

Employee e2 = **new** Employee(200, "Vijay", 66000);

System.***out***.println(e1); // toString() method of Employee

System.***out***.println(e2); // toString() method of Employee

}

}

Output:

Alexandar

Employee [id=100, name=Raj, salary=56000.0]

Employee [id=200, name=Vijay, salary=66000.0]

equals & hashCode

equals method is used to compare the object, by default it compares memory address of an object with another object, but you can override to compare the object properties as per your requirement.

hashCode method returns an unique id, by default it returns unique id but you can override to return the unique id of the object as the per your requirement instead of using the default unique id.

Note: Usually you will override both equals & hashCode, these two methods are useful in certain places i.e., Collection Framework.

Employee.java

**package** com.fundamentals;

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

**private** **int** id;

**private** String name;

**private** **double** salary;

**public** Employee(**int** id, String name, **double** salary) {

**super**();

**this**.id = id;

**this**.name = name;

**this**.salary = salary;

}

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **double** getSalary() {

**return** salary;

}

**public** **void** setSalary(**double** salary) {

**this**.salary = salary;

}

@Override

**public** String toString() {

**return** "Employee [id=" + id + ", name=" + name + ", salary=" + salary + "]";

}

@Override

**public** **int** hashCode() {

**final** **int** prime = 31;

**int** result = 1;

result = prime \* result + id;

**return** result;

}

@Override

**public** **boolean** equals(Object obj) {

**if** (**this** == obj)

**return** **true**;

**if** (obj == **null**)

**return** **false**;

**if** (getClass() != obj.getClass())

**return** **false**;

Employee other = (Employee) obj;

**if** (id != other.id)

**return** **false**;

**return** **true**;

}

}

TestEqualsHashCode.java

**package** com.fundamentals;

**public** **class** TestEqualsHashCode {

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

Employee emp1 = **new** Employee(100, "Raj", 35000);

Employee emp2 = **new** Employee(200, "Bruce", 45000);

Employee emp3 = **new** Employee(100, "Alex", 95000);

System.***out***.println("Equality: "+emp1.equals(emp2)); // false

System.***out***.println("Equality: "+emp1.equals(emp3)); // true

System.***out***.println("Equality: "+emp2.equals(emp3)); // false

System.***out***.println("HashCode: "+emp1.hashCode());

System.***out***.println("HashCode: "+emp2.hashCode());

System.***out***.println("HashCode: "+emp3.hashCode());

}

}

Output:

Equality: false

Equality: true

Equality: false

HashCode: 131

HashCode: 231

HashCode: 131

Wrapper classes:

These classes are present in java.lang, they represent primitive types as object, i.e., every primitive type has corresponding wrapper classes

1. int -> Integer
2. long -> Long
3. short -> Short
4. char -> Character
5. float -> Float
6. double -> Double

int, long, short, char, float, double & etc are all primitive types and you can only do arithmetic operations from them, but with wrapper classes you get methods from it, using them you can perform some extra operations like converting number to string and vice versa, numbers to other number formats like hexadecimal, binary, octal and so on.

TestWrapperClass.java

**package** com.fundamentals;

**public** **class** TestWrapperClass {

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

String amount = "15000.0";

String balance = "25000.0";

**double** amountDouble = Double.*parseDouble*(amount);

**double** balanceDouble = Double.*parseDouble*(balance);

balanceDouble = amountDouble + balanceDouble;

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

// converting int to binary value

**int** intValue = 5;

System.***out***.println("Binary of 5: "+Integer.*toBinaryString*(intValue));

}

}

/\*

Amount : 10000 =====> read that 10000 =====> destination account

\*/

Output:

Balance: 40000.0

Binary of 5: 101

java.util package: It has utility related classes

* Scanner
* ArrayList
* HashSet
* LinkedHashSet
* TreeSet
* Comparator
* Comparable
* HashMap
* LinkedHashMap
* TreeMap
* List
* Set
* Map

Scanner: It is a class used to take input from the keyboard, you can read numbers like int, double, long, read strings, and so on

Scanner sc = new Scanner(System.in);  
sc.nextInt(); // to read int  
sc.nextLong(); // to read Long  
sc.next(); // to read a string  
sc.nextLine(); // to read input till you hit enter

TestScanner.java

**package** com.utility;

**import** java.util.Scanner;

**public** **class** TestScanner {

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

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter a number");

**int** num = scan.nextInt();

System.***out***.println("Enter a string");

String name = scan.next();

System.***out***.println("Enter a double");

**double** salary = scan.nextDouble();

System.***out***.println("Num: "+num+", Name: "+name+", Salary: "+salary);

}

}