**Array: Basic example**

**package** arrays;

**public** **class** ArrayExample {

}

// Pass array as object in method.

/\* public static int getSum(int nums[]) {

int sum = 0;

for(int val : nums)

//sum = sum + val;

sum += val;

return sum;

}

public static void main (String[] args) {

int values[] = {45,76,34,78,223};

int total = getSum(values);

System.out.println(total);

}

}\*/

// Basic of array

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

int values[] = {45,76,34,78,223};

int size = values.length;

for (int index =0; index<size; index++) {

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

}

for(int val : values)

//System.out.println(values[val]);

System.out.println(values);

}

}

\*/

**Another Example of Array:**

**package** arrays;

**import** dryfruit.Fruit;

**public** **class** ArrayExampleObject {

**private** **static** **void** showFruitNames(Fruit fruitData[]) {

**for** (Fruit fr :fruitData ) {

System.***out***.println(fr.getName().toUpperCase());

}

}

**private** **static** String[] getFruitNames(Fruit fruitData[]) {

**int** size = fruitData.length;

String fruitNames[] = **new** String[size];

**int** index = 0;

**for**( Fruit fr : fruitData) {

String name = fr.getName().toUpperCase();

fruitNames[index] = name;

index++;

}

**return** fruitNames;

}

**public** **static** String[] getfruitcolors(Fruit getfrcolr[]) {

**int** size = getfrcolr.length;

String getFruitColors[] = **new** String[size];

**int** index = 0;

**for**(Fruit fr : getfrcolr ) {

String color = fr.getColor().toUpperCase();

getFruitColors[index] = color;

index++;

}

**return** getFruitColors;

}

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

Fruit fruits[] = **new** Fruit[3];

fruits[0] = **new** Fruit();

fruits[1] = **new** Fruit("Apple", "Red");

fruits[2] = **new** Fruit("Orange", "Orange");

String fName[] = *getFruitNames*(fruits);

**for** (String fname : fName)

System.***out***.println(fname);

String fcolor[] = *getfruitcolors*(fruits);

**for** (String fColor : fcolor)

System.***out***.println(fColor);

System.***out***.println(fruits[0].getName());

System.***out***.println(fruits[0].getColor());

System.***out***.println(fruits[1].getName());

System.***out***.println(fruits[1].getColor());

System.***out***.println(fruits[2].getName());

System.***out***.println(fruits[2].getColor());

*showFruitNames*(fruits);

/\* fruits f1 = new Fruit();

fruits f2 = new Fruit("Apple", "Red");

fruits f3 = new Fruit("Orange", "Orange");

Fruit FruitBasket[] = {f1, f2,f3};\*/

}

}

**Fruit Class :**

**package** dryfruit;

**public** **class** Fruit {

**private** String name, color;

**public** Fruit() {

name = "mango";

color = "yellow";

}

**public** Fruit(String name, String color) {

**super**();

**this**.name = name;

**this**.color = color;

}

**public** String getName() {

**return** name;

}

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

**this**.name = name;

}

**public** String getColor() {

**return** color;

}

**public** **void** setColor(String color) {

**this**.color = color;

}

}

**FruitMain :**

**package** dryfruit;

**public** **class** FruitMain {

**public** **static** Fruit changeFruit(Fruit oldFruit) {

oldFruit.setName("Apple");

oldFruit.setColor("Red");

**return** oldFruit;

}

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

Fruit f1 = **new** Fruit();

//Printing value before change.

System.***out***.println(f1.getName());

System.***out***.println(f1.getColor());

Fruit changedFruit = *changeFruit*(f1);

//Printing value after change.

System.***out***.println(changedFruit.getName());

System.***out***.println(changedFruit.getColor());

}

}

/\*

public static void changeFruit(Fruit oldFruit) {

oldFruit.setName("Apple");

oldFruit.setColor("Red");

}

public static void main(String[] args) {

Fruit f1 = new Fruit();

//Printing value before change.

System.out.println(f1.getName());

System.out.println(f1.getColor());

changeFruit(f1);

//Printing value after change.

System.out.println(f1.getName());

System.out.println(f1.getColor());

}

}\*/

**Containment:**

**package** containment;

**public** **class** Contact {

**private** String mobileNo, emailAddress;

**public** Contact() {

mobileNo = "45656564";

emailAddress = "sdsdf@gmail.com";

}

**public** Contact(String mobileNo, String emailAddress) {

**this**.mobileNo = mobileNo;

**this**.emailAddress = emailAddress;

}

**public** String getMobileNo() {

**return** mobileNo;

}

**public** **void** setMobileNo(String mobileNo) {

**this**.mobileNo = mobileNo;

}

**public** String getEmailAddress() {

**return** emailAddress;

}

**public** **void** setEmailAddress(String emailAddress) {

**this**.emailAddress = emailAddress;

}

}

**package** containment;

**public** **class** CreditCard {

**private** String cardNo, bank;

**private** **int** creditLimit;

**public** CreditCard() {

cardNo = "46466";

bank = "HDFC";

creditLimit = 75000;

}

**public** CreditCard(String cardNo, String bank, **int** creditLimit) {

**this**.cardNo = cardNo;

**this**.bank = bank;

**this**.creditLimit = creditLimit;

}

**public** String getCardNo() {

**return** cardNo;

}

**public** **void** setCardNo(String cardNo) {

**this**.cardNo = cardNo;

}

**public** String getBank() {

**return** bank;

}

**public** **void** setBank(String bank) {

**this**.bank = bank;

}

**public** **int** getCreditLimit() {

**return** creditLimit;

}

**public** **void** setCreditLimit(**int** creditLimit) {

**this**.creditLimit = creditLimit;

}

}

**package** containment;

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

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

**private** String name;

**private** Contact contactDetails; // object of another class

**private** CreditCard cardDetails; // object of another class

**public** Customer() {

customerId = 1001;

name = "Dipti";

contactDetails = **new** Contact();

}

**public** Customer(**int** customerId, String name, Contact contactDetails, CreditCard cardDetails) {

**this**.customerId = customerId;

**this**.name = name;

**this**.contactDetails = contactDetails;

**this**.cardDetails = cardDetails;

}

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

**return** customerId;

}

**public** **void** setCustomerId(**int** customerId) {

**this**.customerId = customerId;

}

**public** String getName() {

**return** name;

}

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

**this**.name = name;

}

**public** Contact getContactDetails() {

**return** contactDetails;

}

**public** **void** setContactDetails(Contact contactDetails) {

**this**.contactDetails = contactDetails;

}

**public** CreditCard getCardDetails() {

**return** cardDetails;

}

**public** **void** setCardDetails(CreditCard cardDetails) {

**this**.cardDetails = cardDetails;

}

}

**package** containment;

**public** **class** CustomerMain {

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

// creating customer without credit card.

Customer c1 = **new** Customer();

// Creating a customer with credit card.

Contact contact2 = **new** Contact("4535345", "ere@gmail.com");

CreditCard creditcard2 = **new** CreditCard("5453", "HSBC", 60000);

Customer c2 = **new** Customer(1002, "Bill", contact2, creditcard2);

Contact ct = c1.getContactDetails();

System.***out***.println(ct.getEmailAddress());

String ct2 = c1.getContactDetails().getEmailAddress();

System.***out***.println(ct2);

System.***out***.println(c2.getContactDetails().getEmailAddress());

CreditCard cc = c1.getCardDetails();

**if** (cc != **null**)

System.***out***.println(cc.getCreditLimit());

**else**

System.***out***.println("No Credit card");

}

}

1. **Single Inheritance :**

**package** Inheritence;

**public** **class** Player {

**private** String name;

**private** **int** age;

**public** String getName() {

**return** name;

}

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

**this**.name = name;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

}

**package** Inheritence;

**public** **class** CricketPayer **extends** Player {

**private** **int** runs;

**public** **int** getRuns() {

**return** runs;

}

**public** **void** setRuns(**int** runs) {

**this**.runs = runs;

}

}

**package** Inheritence;

**public** **class** PlayerMain {

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

CricketPayer cp = **new** CricketPayer();

cp.setName("Sachin T");

cp.setAge(55);

cp.setRuns(145366);

}

}

**2. Single Inheritance with constructor: Invoke – down to top , execution top to bottom**

**package** Inheritence;

**public** **class** Player {

**private** String name;

**private** **int** age;

**public** Player() {

name = "Sania";

age = 50;

System.***out***.println("In Player Class");

}

**public** String getName() {

**return** name;

}

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

**this**.name = name;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

}

**package** Inheritence;

**public** **class** CricketPayer **extends** Player {

**private** **int** runs;

**public** CricketPayer() {

runs = 5000;

System.***out***.println("In CricketPlayer Class");

}

**public** **int** getRuns() {

**return** runs;

}

**public** **void** setRuns(**int** runs) {

**this**.runs = runs;

}

}

**package** Inheritence;

**public** **class** PlayerMain {

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

CricketPayer cp = **new** CricketPayer();

/\* cp.setName("Sachin T");

cp.setAge(55);

cp.setRuns(145366);\*/

System.***out***.println(cp.getName());

System.***out***.println(cp.getAge());

System.***out***.println(cp.getRuns());

}

}

**Super Keyword with parameterized constructor:**

**package** Inheritence;

**public** **class** Player {

**private** String name;

**private** **int** age;

**public** Player() {

name = "Sania";

age = 50;

System.***out***.println("In Player Class");

}

**public** Player(String name, **int** age) {

**this**.name = name;

**this**.age = age;

}

**public** Player(**int** age, String name) {

**this**.age = age;

**this**.name = name;

}

**public** String getName() {

**return** name;

}

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

**this**.name = name;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

}

**package** Inheritence;

**public** **class** CricketPayer **extends** Player {

**private** **int** runs;

**public** CricketPayer() {

runs = 5000;

System.***out***.println("In CricketPlayer Class");

}

**public** CricketPayer(String name, **int** age, **int** runs) {

**super**(name, age);

**this**.runs = runs;

}

**public** **int** getRuns() {

**return** runs;

}

**public** **void** setRuns(**int** runs) {

**this**.runs = runs;

}

}

**package** Inheritence;

**public** **class** PlayerMain {

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

CricketPayer cp = **new** CricketPayer();

CricketPayer cp1 = **new** CricketPayer("Dhoni", 50, 34300);

/\* cp.setName("Sachin T");

cp.setAge(55);

cp.setRuns(145366);\*/

System.***out***.println(cp.getName());

System.***out***.println(cp.getAge());

System.***out***.println(cp.getRuns());

System.***out***.println(cp1.getName());

System.***out***.println(cp1.getAge());

System.***out***.println(cp1.getRuns());

}

}

Polymorphism : Method Overriding.

**package** overriding;

**public** **class** Shape {

**private** String name;

**public** Shape() {

name = "shape1";

}

**public** Shape(String name) {

**this**.name = name;

}

**public** String getName() {

**return** name;

}

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

**this**.name = name;

}

**public** **float** getArea() {

**return** 0.0f;

}

}

**package** overriding;

**public** **class** Rectangle **extends** Shape {

**private** **float** lenght, breadth;

**public** Rectangle() {

lenght = 5;

breadth = 6;

}

**public** Rectangle(String name, **float** lenght, **float** breadth) {

**super**(name);

**this**.lenght = lenght;

**this**.breadth = breadth;

}

**public** **float** getLenght() {

**return** lenght;

}

**public** **void** setLenght(**float** lenght) {

**this**.lenght = lenght;

}

**public** **float** getBreadth() {

**return** breadth;

}

**public** **void** setBreadth(**float** breadth) {

**this**.breadth = breadth;

}

**public** **float** getArea() {

**return** lenght \* breadth;

}

}

**package** overriding;

**public** **class** Circle **extends** Shape {

**private** **float** radius;

**public** Circle() {

radius = 4;

}

**public** **float** getRadius() {

**return** radius;

}

**public** **void** setRadius(**float** radius) {

**this**.radius = radius;

}

**public** Circle(String name, **float** radius) {

**super**(name);

**this**.radius = radius;

}

@Override **public** **float** getArea() { // Override is for compiler to check for the override but this is not mandatory.

**return** 3.14f \*radius \* radius;

}

}

**package** overriding;

**public** **class** ShapeMain {

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

Shape shapes[] = **new** Shape[4];

shapes[0] = **new** Rectangle();

shapes[1] = **new** Circle("Shape 2", 5);

shapes[2] = **new** Rectangle("Shape 3", 2, 5);

shapes[3] = **new** Circle("Shape 4", 6);

System.***out***.println("Printing area of all 4 shapes");

**for** (Shape sh : shapes) {

**float** area = sh.getArea();

System.***out***.println(area);

}

}

}

**toString : Used existing class**

**package** containment;

**public** **class** Contact {

**private** String mobileNo, emailAddress;

**public** Contact() {

mobileNo = "45656564";

emailAddress = "sdsdf@gmail.com";

}

**public** Contact(String mobileNo, String emailAddress) {

**this**.mobileNo = mobileNo;

**this**.emailAddress = emailAddress;

}

**public** String getMobileNo() {

**return** mobileNo;

}

**public** **void** setMobileNo(String mobileNo) {

**this**.mobileNo = mobileNo;

}

**public** String getEmailAddress() {

**return** emailAddress;

}

**public** **void** setEmailAddress(String emailAddress) {

**this**.emailAddress = emailAddress;

}

@Override // toString overriding with Object class , if we will not have this method then the print method return in customer main class will return address.

To add this Source – Generate -- toString

**public** String toString() {

**return** "Contact [mobileNo=" + mobileNo + ", emailAddress=" + emailAddress + "]";

}

}

**package** containment;

**public** **class** CreditCard {

**private** String cardNo, bank;

**private** **int** creditLimit;

**public** CreditCard() {

cardNo = "46466";

bank = "HDFC";

creditLimit = 75000;

}

**public** CreditCard(String cardNo, String bank, **int** creditLimit) {

**this**.cardNo = cardNo;

**this**.bank = bank;

**this**.creditLimit = creditLimit;

}

**public** String getCardNo() {

**return** cardNo;

}

**public** **void** setCardNo(String cardNo) {

**this**.cardNo = cardNo;

}

**public** String getBank() {

**return** bank;

}

**public** **void** setBank(String bank) {

**this**.bank = bank;

}

**public** **int** getCreditLimit() {

**return** creditLimit;

}

**public** **void** setCreditLimit(**int** creditLimit) {

**this**.creditLimit = creditLimit;

}

}

**package** containment;

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

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

**private** String name;

**private** Contact contactDetails; // object of another class

**private** CreditCard cardDetails; // object of another class

**public** Customer() {

customerId = 1001;

name = "Dipti";

contactDetails = **new** Contact();

}

**public** Customer(**int** customerId, String name, Contact contactDetails, CreditCard cardDetails) {

**this**.customerId = customerId;

**this**.name = name;

**this**.contactDetails = contactDetails;

**this**.cardDetails = cardDetails;

}

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

**return** customerId;

}

**public** **void** setCustomerId(**int** customerId) {

**this**.customerId = customerId;

}

**public** String getName() {

**return** name;

}

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

**this**.name = name;

}

**public** Contact getContactDetails() {

**return** contactDetails;

}

**public** **void** setContactDetails(Contact contactDetails) {

**this**.contactDetails = contactDetails;

}

**public** CreditCard getCardDetails() {

**return** cardDetails;

}

**public** **void** setCardDetails(CreditCard cardDetails) {

**this**.cardDetails = cardDetails;

}

}

**package** containment;

**public** **class** CustomerMain {

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

// creating customer without credit card.

Customer c1 = **new** Customer();

// Creating a customer with credit card.

Contact contact2 = **new** Contact("4535345", "ere@gmail.com");

CreditCard creditcard2 = **new** CreditCard("5453", "HSBC", 60000);

System.***out***.println(contact2); // this is implicit call, for explicit call System.***out***.println(contact2.toString());

Customer c2 = **new** Customer(1002, "Bill", contact2, creditcard2);

Contact ct = c1.getContactDetails();

System.***out***.println(ct.getEmailAddress());

String ct2 = c1.getContactDetails().getEmailAddress();

System.***out***.println(ct2);

System.***out***.println(c2.getContactDetails().getEmailAddress());

CreditCard cc = c1.getCardDetails();

**if** (cc != **null**)

System.***out***.println(cc.getCreditLimit());

**else**

System.***out***.println("No Credit card");

}

}

**Inherited toString:**

**package** Inheritence;

**public** **class** Player {

**private** String name;

**private** **int** age;

**public** **int** x;

**public** Player() {

name = "Sania";

age = 50;

System.***out***.println("In Player Class");

}

**public** Player(String name, **int** age) {

**this**.name = name;

**this**.age = age;

}

**public** Player(**int** age, String name) {

**this**.age = age;

**this**.name = name;

}

**public** String getName() {

**return** name;

}

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

**this**.name = name;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

@Override

**public** String toString() {

**return** "Player [name=" + name + ", age=" + age + "]";

}

}

**package** Inheritence;

**public** **class** Player {

**private** String name;

**private** **int** age;

**public** **int** x;

**public** Player() {

name = "Sania";

age = 50;

System.***out***.println("In Player Class");

}

**public** Player(String name, **int** age) {

**this**.name = name;

**this**.age = age;

}

**public** Player(**int** age, String name) {

**this**.age = age;

**this**.name = name;

}

**public** String getName() {

**return** name;

}

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

**this**.name = name;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

@Override

**public** String toString() {

**return** "Player [name=" + name + ", age=" + age + "]";

}

}

**package** Inheritence;

**public** **class** PlayerMain {

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

//CricketPayer cp = new CricketPayer();

CricketPayer cp1 = **new** CricketPayer("Dhoni", 50, 34300);

System.***out***.println(cp1);

/\* System.out.println(cp.getName());

System.out.println(cp.getAge());

System.out.println(cp.getRuns());\*/

System.***out***.println(cp1.getName());

System.***out***.println(cp1.getAge());

System.***out***.println(cp1.getRuns());

}

}

Equals(Object)

**package** containment;

**public** **class** Contact {

**private** String mobileNo, emailAddress;

**public** Contact() {

mobileNo = "45656564";

emailAddress = "sdsdf@gmail.com";

}

**public** Contact(String mobileNo, String emailAddress) {

**this**.mobileNo = mobileNo;

**this**.emailAddress = emailAddress;

}

**public** String getMobileNo() {

**return** mobileNo;

}

**public** **void** setMobileNo(String mobileNo) {

**this**.mobileNo = mobileNo;

}

**public** String getEmailAddress() {

**return** emailAddress;

}

**public** **void** setEmailAddress(String emailAddress) {

**this**.emailAddress = emailAddress;

}

@Override

**public** String toString() {

**return** "Contact [mobileNo=" + mobileNo + ", emailAddress=" + emailAddress + "]";

}

@Override

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

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

**int** result = 1;

result = prime \* result + ((emailAddress == **null**) ? 0 : emailAddress.hashCode());

result = prime \* result + ((mobileNo == **null**) ? 0 : mobileNo.hashCode());

**return** result;

}

@Override

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

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

**return** **true**;

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

**return** **false**;

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

**return** **false**;

Contact other = (Contact) obj;

**if** (emailAddress == **null**) {

**if** (other.emailAddress != **null**)

**return** **false**;

} **else** **if** (!emailAddress.equals(other.emailAddress))

**return** **false**;

**if** (mobileNo == **null**) {

**if** (other.mobileNo != **null**)

**return** **false**;

} **else** **if** (!mobileNo.equals(other.mobileNo))

**return** **false**;

**return** **true**;

}

}

**package** containment;

**public** **class** CreditCard {

**private** String cardNo, bank;

**private** **int** creditLimit;

**public** CreditCard() {

cardNo = "46466";

bank = "HDFC";

creditLimit = 75000;

}

**public** CreditCard(String cardNo, String bank, **int** creditLimit) {

**this**.cardNo = cardNo;

**this**.bank = bank;

**this**.creditLimit = creditLimit;

}

**public** String getCardNo() {

**return** cardNo;

}

**public** **void** setCardNo(String cardNo) {

**this**.cardNo = cardNo;

}

**public** String getBank() {

**return** bank;

}

**public** **void** setBank(String bank) {

**this**.bank = bank;

}

**public** **int** getCreditLimit() {

**return** creditLimit;

}

**public** **void** setCreditLimit(**int** creditLimit) {

**this**.creditLimit = creditLimit;

}

}

**package** containment;

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

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

**private** String name;

**private** Contact contactDetails; // object of another class

**private** CreditCard cardDetails; // object of another class

**public** Customer() {

customerId = 1001;

name = "Dipti";

contactDetails = **new** Contact();

}

**public** Customer(**int** customerId, String name, Contact contactDetails, CreditCard cardDetails) {

**this**.customerId = customerId;

**this**.name = name;

**this**.contactDetails = contactDetails;

**this**.cardDetails = cardDetails;

}

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

**return** customerId;

}

**public** **void** setCustomerId(**int** customerId) {

**this**.customerId = customerId;

}

**public** String getName() {

**return** name;

}

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

**this**.name = name;

}

**public** Contact getContactDetails() {

**return** contactDetails;

}

**public** **void** setContactDetails(Contact contactDetails) {

**this**.contactDetails = contactDetails;

}

**public** CreditCard getCardDetails() {

**return** cardDetails;

}

**public** **void** setCardDetails(CreditCard cardDetails) {

**this**.cardDetails = cardDetails;

}

}

**package** containment;

**public** **class** CustomerMain {

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

// creating customer without credit card.

Customer c1 = **new** Customer();

// Creating a customer with credit card.

Contact contact2 = **new** Contact("4535345", "ere@gmail.com");

Contact contact3 = **new** Contact("4535345", "ere@gmail.com");

System.***out***.println(contact2.equals(contact3));

CreditCard creditcard2 = **new** CreditCard("5453", "HSBC", 60000);

//System.out.println(contact2);

Customer c2 = **new** Customer(1002, "Bill", contact2, creditcard2);

Contact ct = c1.getContactDetails();

System.***out***.println(ct.getEmailAddress());

String ct2 = c1.getContactDetails().getEmailAddress();

System.***out***.println(ct2);

System.***out***.println(c2.getContactDetails().getEmailAddress());

CreditCard cc = c1.getCardDetails();

**if** (cc != **null**)

System.***out***.println(cc.getCreditLimit());

**else**

System.***out***.println("No Credit card");

}

}

Abstract Class:

**package** abstract\_classes;

**public** **abstract** **class** MusicalInstrument {

**private** String id;

**private** **int** cost;

**public** **abstract** **void** play();

**public** MusicalInstrument() {

id = "m1";

cost = 15000;

}

**public** MusicalInstrument(String id, **int** cost) {

**this**.id = id;

**this**.cost = cost;

}

**public** String getId() {

**return** id;

}

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

**this**.id = id;

}

**public** **int** getCost() {

**return** cost;

}

**public** **void** setCost(**int** cost) {

**this**.cost = cost;

}

}

**package** abstract\_classes;

**public** **class** Piano **extends** MusicalInstrument {

**private** **int** strings;

**public** Piano() {

strings = 48;

}

**public** Piano(String id, **int** cost, **int** strings) {

**super**(id, cost);

**this**.strings = strings;

}

@Override

**public** **void** play() {

System.***out***.println("playing Piano with " + strings + " strings");

}

}

**package** abstract\_classes;

**public** **class** Tabla **extends** MusicalInstrument {

**private** String surface;

**public** Tabla() {

surface = "Hard";

}

**public** Tabla(String id, **int** cost, String surface) {

**super**(id, cost);

**this**.surface = surface;

}

**public** String getSurface() {

**return** surface;

}

**public** **void** setSurface(String surface) {

**this**.surface = surface;

}

@Override

**public** **void** play() {

System.***out***.println("Playing a table with: " +surface + "surface.");

}

}

**package** abstract\_classes;

**public** **class** MusicalInstrumentMain {

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

MusicalInstrument instruments[] = **new** MusicalInstrument[2];

instruments[0] = **new** Piano();

instruments[1] = **new** Tabla("M2", 7500, "Medium");

**for** (MusicalInstrument ins : instruments)

ins.play();

}

}

**Wrapper/ Argument:**

**public** **class** misc {

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

//program to accept information about a Person.

//as Name , age, weight in the form of command line argument

String name = args[0];

String sAge = args[1];

String sWeight = args[2];

**int** age = Integer.*parseInt*(sAge); // Integer wrapper

**float** weight = Float.*parseFloat*(sWeight);

StringBuffer personData = **new** StringBuffer();

personData.append("Name: ");

personData.append(name);

personData.append("Age: ");

personData.append(age);

personData.append("Weight: ");

personData.append(weight);

System.***out***.println(personData);

}

}

**To run above code , run configuration , select your class in main and then pass arguments. After this apply and run.**

**Interface :**

**package** interfaces;

**public** **interface** GreetingService {

String greet(String userName);

}

**package** interfaces;

**public** **class** HelloService **implements** GreetingService {

@Override

**public** String greet(String user) {

String message = "Hello " + user;

**return** message;

}

}

**package** interfaces;

**public** **class** WelcomeService **implements** GreetingService {

@Override

**public** String greet(String user) {

String message = "Welcome " + user;

**return** message;

}

}

**package** interfaces;

**public** **class** GreetingServiceMain {

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

String result;

GreetingService gs;

gs = **new** HelloService();

result = gs.greet("Ram");

System.***out***.println(result);

gs = **new** WelcomeService();

result = gs.greet("Shyam");

System.***out***.println(result);

}

}

**Interface example 2 :**

**package** interfaces;

**public** **interface** BisleriBottle {

**int** getVolume();

**int** getCost();

}

**package** interfaces;

**public** **class** BisleriPack {

**private** BisleriBottle bottleType;

**private** **int** quantity;

// Constructors

**public** BisleriPack() {

bottleType = **new** SmallBottle();

quantity = 20;

}

**public** BisleriPack(BisleriBottle bottleType, **int** quantity) {

**this**.bottleType = bottleType;

**this**.quantity = quantity;

}

// Getter - Setter

**public** BisleriBottle getBottleType() {

**return** bottleType;

}

**public** **void** setBottleType(BisleriBottle bottleType) {

**this**.bottleType = bottleType;

}

**public** **int** getQuantity() {

**return** quantity;

}

**public** **void** setQuantity(**int** quantity) {

**this**.quantity = quantity;

}

// Methods

**public** **int** getPackVolume() {

**return** quantity \* bottleType.getVolume();

}

**public** **int** getPackCost() {

**return** quantity \* bottleType.getCost();

}

}

**package** interfaces;

**public** **class** SmallBottle **implements** BisleriBottle {

@Override

**public** **int** getVolume() {

**return** 200;

}

@Override

**public** **int** getCost() {

**return** 5;

}

}

**package** interfaces;

**public** **class** MediumBottle **implements** BisleriBottle {

@Override

**public** **int** getVolume() {

**return** 300;

}

@Override

**public** **int** getCost() {

**return** 8;

}

}

**package** interfaces;

**public** **class** LargeBottle **implements** BisleriBottle {

@Override

**public** **int** getVolume() {

**return** 500;

}

@Override

**public** **int** getCost() {

**return** 10;

}

}

**package** interfaces;

**public** **class** BisleriMain {

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

BisleriPack packs[] = **new** BisleriPack[3];

packs[0] = **new** BisleriPack();

/\* BisleriBottle medium = new MediumBottle();

packs[1] = new BisleriPack(medium, 25);

\*/

BisleriBottle medium = **new** MediumBottle();

packs[1] = **new** BisleriPack(**new** BisleriBottle() {

@Override

**public** **int** getVolume() {

// **TODO** Auto-generated method stub

**return** 0;

}

@Override

**public** **int** getCost() {

// **TODO** Auto-generated method stub

**return** 0;

}

},25);

BisleriBottle large = **new** LargeBottle();

packs[2] = **new** BisleriPack(large, 15);

**int** totalVolume = 0 , totalCost = 0;

**for**(BisleriPack pack : packs) {

totalVolume += pack.getPackVolume();

totalCost += pack.getPackCost();

System.***out***.println(totalVolume);

System.***out***.println(totalCost);

}

}

}

**Design Patter : Singleton**

**package** patterns;

**public** **class** Singleton {

**private** **static** Singleton *ref*;

**private** Singleton() {

}

**static** **public** Singleton createObject() {

**if**(*ref* == **null**)

*ref* = **new** Singleton();

**return** *ref*;

}

}

//Singleton s1 = Singleton.createobject();

//Singleton s2 = Singleton.createobject();

//s1 = s2 = null;

//Singleton s3 = Singleton.createobject();

Error Handling: ArrayIndexOutOfBoundsException, NumberFormatException

**package** StringBuffer;

**public** **class** misc {

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

//program to accept information about a Person.

//as Name , age, weight in the form of command line argument

**try** {

String name = args[0];

String sAge = args[1];

String sWeight = args[2];

**int** age = Integer.*parseInt*(sAge);

**float** weight = Float.*parseFloat*(sWeight);

StringBuffer personData = **new** StringBuffer();

personData.append("Name: ");

personData.append(name);

personData.append("Age: ");

personData.append(age);

personData.append("Weight: ");

personData.append(weight);

System.***out***.println(personData);

}

**catch**(ArrayIndexOutOfBoundsException ex){

System.***out***.println("Enter atleast 3 values");

}

**catch**(NumberFormatException ex) {

System.***out***.println("age and weight should be numeric.");

}

}

}

**Error Handling: one Catch block handling multiple exceptions.**

**package** StringBuffer;

**public** **class** misc {

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

//program to accept information about a Person.

//as Name , age, weight in the form of command line argument

**try** {

String name = args[0];

String sAge = args[1];

String sWeight = args[2];

**int** age = Integer.*parseInt*(sAge);

**float** weight = Float.*parseFloat*(sWeight);

StringBuffer personData = **new** StringBuffer();

personData.append("Name: ");

personData.append(name);

personData.append("Age: ");

personData.append(age);

personData.append("Weight: ");

personData.append(weight);

System.***out***.println(personData);

}

**catch**(ArrayIndexOutOfBoundsException | NumberFormatException ex){ // pipe symbol is a bitwise operator

**if**(ex **instanceof** ArrayIndexOutOfBoundsException)

{

System.***out***.println("Enter atleast 3 values.");

}

**else** **if**(ex **instanceof** NumberFormatException)

{

System.***out***.println("The age or weight is invalid.");

}

}

}

}

**Cascading of exceptions:**

**package** StringBuffer;

**public** **class** misc {

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

//program to accept information about a Person.

//as Name , age, weight in the form of command line argument

**try** {

String name = args[0];

String sAge = args[1];

String sWeight = args[2];

**int** age = Integer.*parseInt*(sAge);

**float** weight = Float.*parseFloat*(sWeight);

StringBuffer personData = **new** StringBuffer();

personData.append("Name: ");

personData.append(name);

personData.append("Age: ");

personData.append(age);

personData.append("Weight: ");

personData.append(weight);

System.***out***.println(personData);

Object obj = personData;

String dataAboutPerson = (String)obj; // Dynamic time of obj is StringBuffer which fails during run time.

System.***out***.println(dataAboutPerson);

}

**catch**(NullPointerException ex) {

System.***out***.println("Null Pointer Exception");

}

**catch**(ArrayIndexOutOfBoundsException | NumberFormatException ex){ // pipe symbol is a bitwise operator

**if**(ex **instanceof** ArrayIndexOutOfBoundsException)

{

System.***out***.println("Enter atleast 3 values.");

}

**else** **if**(ex **instanceof** NumberFormatException)

{

System.***out***.println("The age or weight is invalid.");

}

}

**catch** (Exception ex) {

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

}

**catch** (Throwable ex) {

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

}

}

}

Exception: Throws

**package** Exceptions;

**public** **class** ThrowsMain {

**public** **static** **void** method3() **throws** Exception {

System.***out***.println("In method3");

}

**public** **static** **void** method2() **throws** Exception {

*method3*();

}

**public** **static** **void** method1() **throws** Exception {

*method2*();

}

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

**try** {

*method1*();

} **catch** (Exception e) {

e.printStackTrace(); // It is helpful in debugging the code.

}

}

}

**Another example with unchecked exception**

**package** Exceptions;

**public** **class** ThrowsMain {

**public** **static** **void** method3() **throws** Exception {

System.***out***.println("In method3");

}

**public** **static** **void** method2() **throws** Exception {

*method3*();

}

**public** **static** **void** method1() **throws** Exception {

*method2*();

}

**public** **static** **void** method4() **throws** RuntimeException {

}

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

*method4*(); // Throws in main method or try catch block not mandate for method4 as it is unchecked exception.

**try** {

*method1*();

} **catch** (Exception e) {

e.printStackTrace(); // It is helpful in debugging the code.

}

}

}

**Problem Statement**

**Candidate : name , email address , age (25 to 60)**

**Runtime / Checked**

**InvalidAgeException extends Exception**

**InvalidEmailException extends Exception**

**package** Exceptions;

**public** **class** InvalidAgeException **extends** Exception {

**private** **int** invalidAge;

**public** InvalidAgeException(String errorMessage, **int** invalidAge) {

**super**(errorMessage);

**this**.invalidAge = invalidAge;

}

@Override

**public** String getMessage() {

String msg = **super**.getMessage();

**return** msg + ": " + invalidAge;

}

}

**package** Exceptions;

**public** **class** InvalidEmailException **extends** Exception {

**public** String invalidEmail;

**public** InvalidEmailException(String errormessage, String invalidEmail) {

**super**(errormessage);

**this**.invalidEmail = invalidEmail;

}

@Override

**public** String getMessage() {

String msg = **super**.getMessage();

**return** msg + ": " + invalidEmail;

}

}

**package** Exceptions;

**import** java.util.regex.Pattern;

**public** **class** Candidate {

**private** String name, emailAddress;

**private** **int** age;

**public** Candidate() {

name = "James Gosling";

emailAddress = "james.gosling@oracle.com";

age =59;

}

**public** Candidate(String name, String emailAddress, **int** age) **throws** InvalidAgeException, InvalidEmailException{

**this**.name = name;

**boolean** isAt = emailAddress.contains("@");

**boolean** isDot = emailAddress.contains(".");

**if**(!isAt || !isDot) {

InvalidEmailException ie = **new** InvalidEmailException("Email address is invalid.", emailAddress);

**throw** ie;

}

**this**.emailAddress = emailAddress;

**if**(age < 25 || age >60) {

InvalidAgeException ie = **new** InvalidAgeException("Age is invalid", age);

**throw** ie;

}

**this**.age = age;

}

**public** String getName() {

**return** name;

}

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

**this**.name = name;

}

**public** String getEmailAddress() {

**return** emailAddress;

}

**public** **void** setEmailAddress(String emailAddress) **throws** InvalidEmailException{

**boolean** isAt = emailAddress.contains("@");

**boolean** isDot = emailAddress.contains(".");

**if**(!isAt || !isDot) {

InvalidEmailException ie = **new** InvalidEmailException("Email address is invalid.", emailAddress);

**throw** ie;

}

**this**.emailAddress = emailAddress;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) **throws** InvalidAgeException {

**if**(age < 25 || age >60) {

InvalidAgeException ie = **new** InvalidAgeException("Age is invalid", age);

**throw** ie;

}

**this**.age = age;

}

@Override

**public** String toString() {

**return** "Candidate [name=" + name + ", emailAddress=" + emailAddress + ", age=" + age + "]";

}

}

**package** Exceptions;

**public** **class** CandidateMain {

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

Candidate c1 = **null**, c2 = **null**;

c1 = **new** Candidate();

**try** {

c2 = **new** Candidate("Dipti", "dipti.jain600@gmail.com", 30);

} **catch** (InvalidAgeException | InvalidEmailException e) {

String errMsg = e.getMessage();

System.***out***.println(errMsg);

}

**finally** {

System.***out***.println(c1);

**if**(c2 !=**null**)

System.***out***.println(c2);

}

}

}

**Rethrowing an exception:**

**Lambda Expression :**

**package** interfaces;

@FunctionalInterface

**public** **interface** GreetingService {

String greet(String userName);

}

**package** interfaces;

**public** **class** GreetingServiceMain {

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

GreetingService service1 =

(user) -> "Hello" +user;

GreetingService service2 =

(user) -> "Welcome" +user;

String result = service1.greet("Ram");

System.***out***.println(result);

result = service2.greet("Shyam");

System.***out***.println(result);

/\* String result;

GreetingService gs;

gs = new HelloService();

result = gs.greet("Ram");

System.out.println(result);

gs = new WelcomeService();

result = gs.greet("Shyam");

System.out.println(result);\*/

}

}

**Multithread:**

**1. Extending thread class**

**package** multithreading;

**public** **class** ThreadImp **extends** Thread {

**private** String message;

**private** **int** delayTime;

**public** ThreadImp(String message, **int** delayTime) {

**this**.message = message;

**this**.delayTime = delayTime;

}

**public** **void** run() { // as soon as execution begin it will call run method.

**for** (**int** i =0 ; i<=10; i++) {

System.***out***.println(message);

**try** {

Thread.*sleep*(delayTime);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

}

**package** multithreading;

**public** **class** ThreadMain {

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

Thread t1 = **new** ThreadImp("Hello", 1000); // BORN

Thread t2 = **new** ThreadImp("Welcome", 500);

t1.start(); // Ready t1, if we change t1.start to t1.run then it will behave like main thread only , this will print all hello first and then welcome.

t2.start();

}

}

1. **Implementing runnable interface**

**package** multithreading;

**public** **class** RunnableImpl **implements** Runnable {

**private** String message;

**private** **int** delayTime;

**public** RunnableImpl(String message, **int** delayTime) {

**this**.message = message;

**this**.delayTime = delayTime;

}

@Override

**public** **void** run() {

**for** (**int** i =0 ; i<=10; i++) {

System.***out***.println(message);

**try** {

Thread.*sleep*(delayTime);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

}

**package** multithreading;

**public** **class** RunnableMain{

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

Runnable r1 = **new** RunnableImpl("Hello", 1000);

Runnable r2 = **new** RunnableImpl("Welcome", 500);

Thread t1 = **new** Thread(r1);

Thread t2 = **new** Thread(r2);

t1.start();

t2.start();

}

}

**Join method example :**

**package** multithreading;

**public** **class** RunnableMain{

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

Runnable r1 = **new** RunnableImpl("Hello", 1000);

Runnable r2 = **new** RunnableImpl("Welcome", 500);

Thread t1 = **new** Thread(r1);

Thread t2 = **new** Thread(r2);

t1.start();

t2.start();

**try** {

t2.join();

}

**catch**(Exception e) {

e.printStackTrace();

}

System.***out***.println("Message"); // message will print after threads t2 is dead.

}

}

**package** multithreading;

**public** **class** RunnableImpl **implements** Runnable {

**private** String message;

**private** **int** delayTime;

**public** RunnableImpl(String message, **int** delayTime) {

**this**.message = message;

**this**.delayTime = delayTime;

}

@Override

**public** **void** run() {

**for** (**int** i =0 ; i<=10; i++) {

System.***out***.println(message);

**try** {

Thread.*sleep*(delayTime);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

}

**Multi- threading Synchronization:**

**package** multithreading;

**public** **class** Message {

**private** String greetingMessage;

**public** Message(String greetingMessage) {

**super**();

**this**.greetingMessage = greetingMessage;

}

**Public synchronized** **void** printMessage() **throws** Exception{ // if synchronized is removed from here then it will be Asynchronized.

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

Thread.*sleep*(1000);

System.***out***.println(greetingMessage);

Thread.*sleep*(1000);

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

Thread.*sleep*(1000);

}

}

**package** multithreading;

**public** **class** MessageThread **extends** Thread {

**private** Message messageObject; // Message class object

**public** MessageThread(Message messageObject) {

**super**();

**this**.messageObject = messageObject;

}

**public** **void** run() {

**try** {

messageObject.printMessage();

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

**package** multithreading;

**public** **class** MessageMain {

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

Message m1 = **new** Message("Welcome");

Thread t1 = **new** MessageThread(m1);

Thread t2 = **new** MessageThread(m1);

t1.start();

t2.start();

}

}

**Synchronized Block :**

**package** multithreading;

**public** **class** Message {

**private** String greetingMessage;

**public** Message(String greetingMessage) {

**super**();

**this**.greetingMessage = greetingMessage;

}

**public** **void** printMessage() **throws** Exception{

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

Thread.sleep(1000);

System.out.println(greetingMessage);

Thread.sleep(500);

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

Thread.sleep(1000);

}

}

**package** multithreading;

**public** **class** MessageThread **extends** Thread {

**private** Message messageObject; // Message class object

**public** MessageThread(Message messageObject) {

**super**();

**this**.messageObject = messageObject;

}

**public** **void** run() {

**synchronized** (messageObject) {

**try** {

messageObject.printMessage();

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

}

**package** multithreading;

**public** **class** MessageMain {

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

Message m1 = **new** Message("Welcome");

Thread t1 = **new** MessageThread(m1);

Thread t2 = **new** MessageThread(m1);

t1.start();

t2.start();

}

}

**IO Programming :**

**package** io\_programming;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.io.IOException;

**import** java.io.InputStream;

**public** **class** FileReadMain {

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

InputStream in = **null**;

String filename = "C:\\Users\\adm\\Documents\\DJ57741\\greeting.txt";

**try** {

in = **new** FileInputStream(filename);

**while**(**true**)

{

**int** data = in.read(); // read method throws IOException so had to add try catch block.

**if**(data == -1) {

**break**;

}

System.***out***.print((**char**)data);

}

} **catch** (FileNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} // InputStream is an abstract class so cannot create object so will create ojc/ref of super type.

**catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**finally** {

**try** {

in.close();

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

}

**Try with resources : so no need to close the resources.**

**package io\_programming;**

**import java.io.FileInputStream;**

**import java.io.FileNotFoundException;**

**import java.io.IOException;**

**import java.io.InputStream;**

**public class FileReadMain {**

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

**String filename = "C:\\Users\\adm\\Documents\\DJ57741\\greeting.txt";**

**try (InputStream in = new FileInputStream(filename)){**

**while(true)**

**{**

**int data = in.read(); // read method throws IOException so had to add try catch block.**

**if(data == -1) {**

**break;**

**}**

**System.out.print((char)data);**

**}**

**} catch (FileNotFoundException e) {**

**// TODO Auto-generated catch block**

**e.printStackTrace();**

**} // InputStream is an abstract class so cannot create object so will create ojc/ref of super type.**

**catch (IOException e) {**

**// TODO Auto-generated catch block**

**e.printStackTrace();**

**}**

**}**

**}**

**Buffered Input Stream :**

**package** io\_programming;

**import** java.io.BufferedInputStream;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.io.IOException;

**import** java.io.InputStream;

**public** **class** FileReadMain {

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

String filename = "C:\\Users\\adm\\Documents\\DJ57741\\greeting.txt";

**try** (InputStream in = **new** FileInputStream(filename);

InputStream bin = **new** BufferedInputStream(in)){; // if we want to pass size then (in, size(bytes))

**while**(**true**)

{

**int** data = bin.read(); // read method throws IOException so had to add try catch block.

**if**(data == -1) {

**break**;

}

System.***out***.print((**char**)data);

}

} **catch** (FileNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} // InputStream is an abstract class so cannot create object so will create ojc/ref of super type.

**catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

**Write into file :**

**package** io\_programming;

**import** java.io.BufferedOutputStream;

**import** java.io.FileOutputStream;

**import** java.io.OutputStream;

**public** **class** FileWriteMain {

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

String java = "Java is platform independent language.";

**try**(OutputStream fout = **new** FileOutputStream("java.txt");

OutputStream bout= **new** BufferedOutputStream(fout)){

**byte** data[] = java.getBytes();

bout.write(data); // write first to buffer and when buffered is closed it will write to main file.

}

**catch**(Exception e) {

e.printStackTrace();

}

}

}

**Write File with append data**

**package** io\_programming;

**import** java.io.BufferedOutputStream;

**import** java.io.FileOutputStream;

**import** java.io.OutputStream;

**public** **class** FileWriteMain {

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

String java = "Java is platform independent language. - changed";

**try**(OutputStream fout = **new** FileOutputStream("java.txt", **true**); // true appended data here.

OutputStream bout= **new** BufferedOutputStream(fout)){

**byte** data[] = java.getBytes();

bout.write(data); // write first to buffer and when buffered is closed it will write to main file.

}

**catch**(Exception e) {

e.printStackTrace();

}

}

}

**Random Access File :**

**package** io\_programming;

**import** java.io.RandomAccessFile;

**public** **class** RandomAccessFileMain {

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

String filename = "greeting.txt";

**try**(RandomAccessFile rf = **new** RandomAccessFile(filename, "r")){ // r is for read and w is for right.

**long** size = rf.length();

**long** position = size/2;

rf.seek(size - position);

**byte** data[] = **new** **byte**[(**int**) position];

rf.read(data);

String contents = **new** String (data);

System.***out***.println(contents);

}

**catch**(Exception e) {

e.printStackTrace();

}

}

}

**Buffered Reader:**

**package** io\_programming;

**import** java.io.BufferedReader;

**import** java.io.File;

**import** java.io.FileReader;

**import** java.io.Reader;

**public** **class** BufferReaderMain {

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

String filename = "java.txt";

File currentFile = **new** File(filename);

**if**(currentFile.exists()) {

**try**(Reader fr = **new** FileReader(filename);

BufferedReader br = **new** BufferedReader(fr)){

**while**(**true**) {

String line = br.readLine();

**if**(line == **null**) {

**break**;

}

System.***out***.println(line);

}

}

**catch**(Exception ex) {

ex.printStackTrace();

}

}

**else** {

System.***out***.println("File does not exists.");

}

}

}

**Scanner class :**

**package** io\_programming;

**import** java.util.Scanner;

**public** **class** UserInputMain {

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

**try**(Scanner scr = **new** Scanner(System.***in***)){

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

**int** empno = scr.nextInt();

scr.nextLine();

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

String empName = scr.nextLine();

System.***out***.println("Emp No: " +empno);

System.***out***.println("Name: " +empName );

}

**catch**(Exception e) {

e.printStackTrace();

}

}

}

**Another way to write above program :**

**package** io\_programming;

**import** java.io.IOException;

**import** java.io.InputStream;

**import** java.util.Scanner;

**public** **class** UserInputMain {

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

InputStream is = System.***in***;

**try**( Scanner scr = **new** Scanner(is)){

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

**int** empno = scr.nextInt();

scr.nextLine();

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

String empName = scr.nextLine();

System.***out***.println("Enter Salary: ");

**float** sal = scr.nextFloat();

System.***out***.println("Emp No: " +empno);

System.***out***.println("Name: " +empName );

System.***out***.println("Salary: " +sal );

}

**catch**(Exception e) {

e.printStackTrace();

}

}

}

Serializable :

**package** io\_programming;

**import** java.io.Serializable;

**public** **class** Doctor **implements** Serializable {

// Method will not define thT'S WHY we will call it as markup interface.

**private** String name, specialization;

**public** Doctor() {

**this**.name = "Mr. Tushar";

**this**.specialization = "Dentist";

}

**public** Doctor(String name, String specialization) {

**this**.name = name;

**this**.specialization = specialization;

}

**public** String getName() {

**return** name;

}

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

**this**.name = name;

}

**public** String getSpecialization() {

**return** specialization;

}

**public** **void** setSpecialization(String specialization) {

**this**.specialization = specialization;

}

@Override

**public** String toString() {

**return** "Doctor [name=" + name + ", specialization=" + specialization + "]";

}

}

**package** io\_programming;

**import** java.io.FileOutputStream;

**import** java.io.ObjectOutputStream;

**import** java.io.OutputStream;

**public** **class** SerializationMain {

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

String filename = "Doctor.txt";

**try**(OutputStream fout = **new** FileOutputStream(filename);

ObjectOutputStream out = **new** ObjectOutputStream(fout)){

Doctor doc = **new** Doctor();

out.writeObject(doc);

System.***out***.println("Doctor object serialized");

}

**catch**(Exception e) {

e.printStackTrace();

}

}

}

**package** io\_programming;

**import** java.io.FileInputStream;

**import** java.io.InputStream;

**import** java.io.ObjectInputStream;

**public** **class** DeserializationMain {

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

String filename = "Doctor.txt";

**try**(InputStream is = **new** FileInputStream(filename);

ObjectInputStream oi = **new** ObjectInputStream(is)){

Doctor doc = **new** Doctor();

Object obj = oi.readObject();

System.***out***.println(obj);

}

**catch** (Exception e) {

e.printStackTrace();

}

}

}

**Colections :**

1. **Vector:**

**package** collections;

**import** java.util.Vector;

**import** io\_programming.Doctor;

**public** **class** VectorMain {

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

Vector values = **new** Vector();

values.add("Welcome");

values.add(**new** Doctor("Dr. Tushar", "Dentist"));

values.add(100); // add(new Integer(100))

values.add(**true**);

values.add(45.33);

**int** size = values.size();

**for**(**int** i=0; i<size; i++) {

System.***out***.println(values.get(i));

}

}

}

1. **ArrayList**:

**package** collections;

**import** java.util.ArrayList;

**import** java.util.Vector;

**import** dryfruit.Fruit;

**import** io\_programming.Doctor;

**public** **class** VectorMain {

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

//Vector values = new Vector();

ArrayList values = **new** ArrayList();

values.add("Welcome");

values.add(**new** Doctor("Dr. Tushar", "Dentist"));

values.add(100); // add(new Integer(100))

values.add(**true**);

values.add(45.33);

values.add(100);

values.add("Welcome");

**int** size = values.size();

**for**(**int** i=0; i<size; i++) {

System.***out***.println(values.get(i));

}

}

}

1. **LinkedList**

**package** collections;

**import** java.util.ArrayList;

**import** java.util.LinkedList;

**import** java.util.Vector;

**import** dryfruit.Fruit;

**import** io\_programming.Doctor;

**public** **class** VectorMain {

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

//Vector values = new Vector();

LinkedList values = **new** LinkedList();

values.add("Welcome");

values.add(**new** Doctor("Dr. Tushar", "Dentist"));

values.add(100); // add(new Integer(100))

values.add(**true**);

values.add(45.33);

values.add(100);

values.add("Welcome");

values.addFirst("Hello");

**int** size = values.size();

/\* **for**(**int** i=0; i<size; i++) {

System.***out***.println(values.get(i));

} \*/

**for**(Object obj : values) {

System.***out***.println(obj);

}

System.***out***.println(values.getFirst());

System.***out***.println(values.getLast());

System.***out***.println(values.remove(2));

}

}

1. **Iterating over collections: Iterator**

**package** collections;

**import** java.util.ArrayList;

**import** java.util.LinkedList;

**import** java.util.Vector;

**import** java.util.\*;

**import** io\_programming.Doctor;

**public** **class** VectorMain {

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

//Vector values = new Vector();

LinkedList values = **new** LinkedList();

values.add("Welcome");

values.add(**new** Doctor("Dr. Tushar", "Dentist"));

values.add(100); // add(new Integer(100))

values.add(**true**);

values.add(45.33);

values.add(100);

values.addFirst("Hello");

**int** size = values.size();

/\* for(int i=0; i<size; i++) {

System.out.println(values.get(i));

}\*/

/\* for(Object obj : values) {

System.out.println(obj);

}\*/

Iterator vi = values.iterator();

**while**(vi.hasNext()) {

Object obj = vi.next();

System.***out***.println(obj);

}

}

}

1. **Iterating over collections: Enumeration**
2. **Generic:**

**package** collections;

**public** **interface** Adder<T> {

T add(T t1, T t2);

}

**package** collections;

**public** **class** StringAdder **implements** Adder<String> {

@Override

**public** String add(String s1, String s2) {

String s3 = s1+s2;

**return** s3.toUpperCase();

}

}

**package** collections;

**public** **class** IntAdder **implements** Adder<Integer> {

@Override

**public** Integer add(Integer s1, Integer s2) {

**return** s1+s2;

}

}

**package** collections;

**public** **class** GenericMain {

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

Adder<String> a1 = **new** StringAdder();

Adder<Integer> a2 = **new** IntAdder();

System.***out***.println(a1.add("Dipti", "Jain"));

System.***out***.println(a2.add(12, 21));

}

}

1. **Generic with other example**

**package** collections;

**import** java.util.\*;

**public** **class** CollectUsingGenericsMain {

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

List countries = **new** ArrayList ();

countries.add("India");

countries.add("Germany");

countries.add("usa");

countries.add("japan");

countries.add("uk");

**for**(Object obj : countries) {

String country = (String)obj; // type cast

System.***out***.println(country.toUpperCase());

}

}

}

**But above example is not type safe , to make it generic and type safe :**

**package** collections;

**import** java.util.\*;

**public** **class** CollectUsingGenericsMain {

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

List<String> countries = **new** ArrayList<String>();

countries.add("India");

countries.add("Germany");

countries.add("usa");

countries.add("japan");

countries.add("uk");

**for**(String obj : countries) {

System.***out***.println(obj.toUpperCase());

}

}

}

1. **Set :**

**package** collections;

**import** java.util.HashSet;

**import** java.util.\*;

**public** **class** SetMain {

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

Set<String> flowers = **new** HashSet<>();

flowers.add("Rose");

flowers.add("lily");

flowers.add("champa");

flowers.add("jasmine");

flowers.add("Lotus");

flowers.add("Rose");

flowers.add("Lotus");

Iterator<String> it = flowers.iterator();

**while**(it.hasNext()) {

String flo = it.next();

System.***out***.println(flo.toUpperCase());

}

/\* for (String flo1 : flowers) {

System.out.println(flo1);

}\*/

}

}

1. **HashSet**

**package** collections;

**import** java.util.Objects;

**public** **class** Flower {

**private** String name;

**private** String color;

**public** Flower() {

**this**.name = "Lotus";

**this**.color = "pink";

}

**public** Flower(String name, String color) {

**this**.name = name;

**this**.color = color;

}

**public** String getName() {

**return** name;

}

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

**this**.name = name;

}

**public** String getColor() {

**return** color;

}

**public** **void** setColor(String color) {

**this**.color = color;

}

@Override

**public** String toString() {

**return** "Flower [name=" + name + ", color=" + color + "]";

}

@Override

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

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

**int** result = 1;

result = prime \* result + ((color == **null**) ? 0 : color.hashCode());

result = prime \* result + ((name == **null**) ? 0 : name.hashCode());

**return** result;

}

@Override

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

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

**return** **true**;

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

**return** **false**;

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

**return** **false**;

Flower other = (Flower) obj;

**if** (color == **null**) {

**if** (other.color != **null**)

**return** **false**;

} **else** **if** (!color.equals(other.color))

**return** **false**;

**if** (name == **null**) {

**if** (other.name != **null**)

**return** **false**;

} **else** **if** (!name.equals(other.name))

**return** **false**;

**return** **true**;

}

}

**package** collections;

**import** java.util.HashSet;

**import** java.util.Set;

**public** **class** FlowerSetMain {

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

Set<Flower> flowers = **new** HashSet<>();

flowers.add(**new** Flower());

flowers.add(**new** Flower("Rose", "Red"));

flowers.add(**new** Flower("Lily", "white"));

flowers.add(**new** Flower("Rose", "Red"));

flowers.add(**new** Flower("Habiscus", "Red"));

**for**(Flower flower : flowers) {

System.***out***.println(flower);

}

}

}

Another example of hashset

**package** collections;

**import** java.util.Objects;

**public** **class** Flower {

**private** String name;

**private** String color;

**public** Flower() {

**this**.name = "Lotus";

**this**.color = "pink";

}

**public** Flower(String name, String color) {

**this**.name = name;

**this**.color = color;

}

**public** String getName() {

**return** name;

}

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

**this**.name = name;

}

**public** String getColor() {

**return** color;

}

**public** **void** setColor(String color) {

**this**.color = color;

}

@Override

**public** String toString() {

**return** "Flower [name=" + name + ", color=" + color + "]";

}

@Override

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

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

**int** result = 1;

result = prime \* result + ((color == **null**) ? 0 : color.hashCode());

result = prime \* result + ((name == **null**) ? 0 : name.hashCode());

**return** result;

}

@Override

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

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

**return** **true**;

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

**return** **false**;

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

**return** **false**;

Flower other = (Flower) obj;

**if** (color == **null**) {

**if** (other.color != **null**)

**return** **false**;

} **else** **if** (!color.equals(other.color))

**return** **false**;

**if** (name == **null**) {

**if** (other.name != **null**)

**return** **false**;

} **else** **if** (!name.equals(other.name))

**return** **false**;

**return** **true**;

}

}

**package** collections;

**import** java.util.HashSet;

**import** java.util.Set;

**public** **class** FlowerSetMain {

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

Set<Flower> flowers = **new** HashSet<>();

flowers.add(**new** Flower());

flowers.add(**new** Flower("Rose", "Red"));

flowers.add(**new** Flower("Lily", "white"));

flowers.add(**new** Flower("Rose", "Red"));

flowers.add(**new** Flower("Habiscus", "Red"));

**for**(Flower flower : flowers)

System.***out***.println(flower);

Flower f1 = **new** Flower("SunFlower", "Yellow");

Flower f2 = **new** Flower("Lily", "white");

**boolean** found = flowers.contains(f1);

System.***out***.println(found);

found = flowers.contains(f2);

System.***out***.println(found);

}

}

**Map :**

**package** collections;

**import** java.util.HashMap;

**import** java.util.Map;

**import** java.util.Set;

**public** **class** MapMain {

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

Map<String, Flower> flowers = **new** HashMap<>();

flowers.put("f1", **new** Flower());

flowers.put("f2", **new** Flower("Jasmine", "Red"));

flowers.put("f3", **new** Flower("Rose", "Red"));

flowers.put("f4", **new** Flower("Lilly", "White"));

Set<String> keys = flowers.keySet(); // this will return key-set

**for**(String key : keys) {

Flower flower = flowers.get(key); // getting value against key.

System.***out***.println(key + " " +flower); }

}

}

**package** collections;

**import** java.util.Objects;

**public** **class** Flower {

**private** String name;

**private** String color;

**public** Flower() {

**this**.name = "Lotus";

**this**.color = "pink";

}

**public** Flower(String name, String color) {

**this**.name = name;

**this**.color = color;

}

**public** String getName() {

**return** name;

}

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

**this**.name = name;

}

**public** String getColor() {

**return** color;

}

**public** **void** setColor(String color) {

**this**.color = color;

}

@Override

**public** String toString() {

**return** "Flower [name=" + name + ", color=" + color + "]";

}

@Override

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

**int** lenght = name.length() + color.length();

**return** lenght;

/\* final int prime = 31;

int result = 1;

result = prime \* result + ((color == null) ? 0 : color.hashCode());

result = prime \* result + ((name == null) ? 0 : name.hashCode());

return result;\*/

}

@Override

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

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

**return** **true**;

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

**return** **false**;

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

**return** **false**;

Flower other = (Flower) obj;

**if** (color == **null**) {

**if** (other.color != **null**)

**return** **false**;

} **else** **if** (!color.equals(other.color))

**return** **false**;

**if** (name == **null**) {

**if** (other.name != **null**)

**return** **false**;

} **else** **if** (!name.equals(other.name))

**return** **false**;

**return** **true**;

}

}

**Getting only value using below update in MapMain:**

**package** collections;

**import** java.util.Collection;

**import** java.util.HashMap;

**import** java.util.Map;

**import** java.util.Set;

**public** **class** MapMain {

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

Map<String, Flower> flowers = **new** HashMap<>();

flowers.put("f1", **new** Flower());

flowers.put("f2", **new** Flower("Jasmine", "Red"));

flowers.put("f3", **new** Flower("Rose", "Red"));

flowers.put("f4", **new** Flower("Lilly", "White"));

Set<String> keys = flowers.keySet(); // this will return key-set

**for**(String key : keys) {

Flower flower = flowers.get(key); // getting value against key.

System.***out***.println(key + " " +flower);

}

Collection<Flower> flowervals = flowers.values();

**for**(Flower f1 : flowervals)

System.***out***.println(f1);

}

}

**Properties :**

**Windows.properties file :**

title=Welcome Dipti

width=500

height=400

**package** collections;

**import** java.io.FileInputStream;

**import** java.io.InputStream;

**import** java.util.Enumeration;

**import** java.util.Properties;

**import** javax.swing.JFrame;

**public** **class** WindowMain {

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

Properties windowProps = **new** Properties();

String filename = "window.properties";

**try**(InputStream fin = **new** FileInputStream(filename)){

windowProps.load(fin);

String title = windowProps.getProperty("title");

String width = windowProps.getProperty("width");

**int** wid = Integer.*parseInt*(width);

String height = windowProps.getProperty("height");

**int** hght = Integer.*parseInt*(height);

Enumeration names = windowProps.propertyNames();

**while**(names.hasMoreElements()) {

Object name = names.nextElement();

Object value = windowProps.get(name);

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

}

//Code to display a window

JFrame frame = **new** JFrame(title);

frame.setSize(wid, hght);

frame.setVisible(**true**);

frame.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

}

**catch**(Exception e) {

e.printStackTrace();

}

}

}

**SortedSet :**

**package** collections;

**public** **class** Flower **implements** Comparable<Flower> {

**private** String name;

**private** String color;

**public** Flower() {

**this**.name = "Lotus";

**this**.color = "pink";

}

**public** Flower(String name, String color) {

**this**.name = name;

**this**.color = color;

}

**public** String getName() {

**return** name;

}

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

**this**.name = name;

}

**public** String getColor() {

**return** color;

}

**public** **void** setColor(String color) {

**this**.color = color;

}

@Override

**public** String toString() {

**return** "Flower [name=" + name + ", color=" + color + "]";

}

@Override

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

**int** lenght = name.length() + color.length();

**return** lenght;

/\* final int prime = 31;

int result = 1;

result = prime \* result + ((color == null) ? 0 : color.hashCode());

result = prime \* result + ((name == null) ? 0 : name.hashCode());

return result;\*/

}

@Override

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

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

**return** **true**;

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

**return** **false**;

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

**return** **false**;

Flower other = (Flower) obj;

**if** (color == **null**) {

**if** (other.color != **null**)

**return** **false**;

} **else** **if** (!color.equals(other.color))

**return** **false**;

**if** (name == **null**) {

**if** (other.name != **null**)

**return** **false**;

} **else** **if** (!name.equals(other.name))

**return** **false**;

**return** **true**;

}

@Override

**public** **int** compareTo(Flower flower2) {

String n1 = name; // this.name

String n2 = flower2.name;

**int** comparision = n1.compareTo(n2);

**if**(comparision == 0)

comparision ++;

**return** comparision;

}

}

**package** collections;

**import** java.util.SortedSet;

**import** java.util.TreeSet;

**public** **class** SortedSetFlowerMain {

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

SortedSet<Flower> flowers = **new** TreeSet<>();

flowers.add(**new** Flower());

flowers.add(**new** Flower("Rose", "Red"));

flowers.add(**new** Flower("Lily", "white"));

flowers.add(**new** Flower("Rose", "Red"));

flowers.add(**new** Flower("Habiscus", "Red"));

**for**(Flower flower : flowers)

System.***out***.println(flower);

}

}

**Comparable with comparator :**

**package** collections;

**public** **class** Flower **implements** Comparable<Flower> {

**private** String name;

**private** String color;

**public** Flower() {

**this**.name = "Lotus";

**this**.color = "pink";

}

**public** Flower(String name, String color) {

**this**.name = name;

**this**.color = color;

}

**public** String getName() {

**return** name;

}

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

**this**.name = name;

}

**public** String getColor() {

**return** color;

}

**public** **void** setColor(String color) {

**this**.color = color;

}

@Override

**public** String toString() {

**return** "Flower [name=" + name + ", color=" + color + "]";

}

@Override

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

**int** lenght = name.length() + color.length();

**return** lenght;

/\* final int prime = 31;

int result = 1;

result = prime \* result + ((color == null) ? 0 : color.hashCode());

result = prime \* result + ((name == null) ? 0 : name.hashCode());

return result;\*/

}

@Override

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

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

**return** **true**;

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

**return** **false**;

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

**return** **false**;

Flower other = (Flower) obj;

**if** (color == **null**) {

**if** (other.color != **null**)

**return** **false**;

} **else** **if** (!color.equals(other.color))

**return** **false**;

**if** (name == **null**) {

**if** (other.name != **null**)

**return** **false**;

} **else** **if** (!name.equals(other.name))

**return** **false**;

**return** **true**;

}

@Override

**public** **int** compareTo(Flower flower2) {

String n1 = name; // this.name

String n2 = flower2.name;

**int** comparision = n1.compareTo(n2);

**if**(comparision == 0)

comparision ++;

**return** comparision;

}

}

**package** collections;

**import** java.util.Comparator;

**public** **class** FlowerSortColorAsc **implements** Comparator<Flower> {

@Override

**public** **int** compare(Flower f1, Flower f2) {

String c1 = f1.getColor();

String c2 = f2.getColor();

**int** comparison = c1.compareTo(c2);

**if**(comparison == 0)

comparison++;

**return** comparison;

}

}

**package** collections;

**import** java.util.Comparator;

**import** java.util.SortedSet;

**import** java.util.TreeSet;

**public** **class** SortedSetFlowerMain {

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

Comparator<Flower> comp1 = **new** FlowerSortColorAsc();

Comparator<Flower> comp2 = (f1,f2) -> {

String c1 = f1.getColor();

String c2 = f2.getColor();

**int** comparison = c2.compareTo(c1);

**if**(comparison == 0)

comparison++;

**return** comparison;

};

SortedSet<Flower> flowers = **new** TreeSet<>(comp2); // Comp1 or pass comp2 here will sort color in asc or desc otherwise if left blank then will get name asc which is default due to comparable.

flowers.add(**new** Flower());

flowers.add(**new** Flower("Rose", "yellow"));

flowers.add(**new** Flower("Lily", "white"));

flowers.add(**new** Flower("Rose", "red"));

flowers.add(**new** Flower("Habiscus", "red"));

**for**(Flower flower : flowers)

System.***out***.println(flower);

}

}

**Networking :**

**package** networking;

**import** java.io.DataInputStream;

**import** java.io.DataOutputStream;

**import** java.io.InputStream;

**import** java.io.OutputStream;

**import** java.net.InetAddress;

**import** java.net.Socket;

**import** java.util.Scanner;

**public** **class** Client {

**private** Socket csocket;

**private** DataOutputStream dout;

**private** DataInputStream din; // Data will be in form of String will be passed in format Unicode String format.

**public** Client() **throws** Exception {

InetAddress ipAddress = InetAddress.*getLocalHost*();

**int** portNo = 3000;

csocket = **new** Socket(ipAddress, portNo);

OutputStream out = csocket.getOutputStream();

InputStream in = csocket.getInputStream();

dout = **new** DataOutputStream(out);

din = **new** DataInputStream(in);

}

**public** **void** startChat() **throws** Exception{

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

**while**(**true**) {

String data = scr.nextLine();

dout.writeUTF(data);

data = din.readUTF();

System.***out***.println(data);

}

}

}

**package** networking;

**public** **class** ClientMain {

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

**try** {

Client clientApp = **new** Client();

clientApp.startChat();

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

**package** networking;

**import** java.io.DataInputStream;

**import** java.io.DataOutputStream;

**import** java.io.InputStream;

**import** java.io.OutputStream;

**import** java.net.ServerSocket;

**import** java.net.Socket;

**import** java.util.Scanner;

**public** **class** Server {

**private** ServerSocket ssSocket;

**public** Server() **throws** Exception {

ssSocket = **new** ServerSocket(3000, 6); // this is like server is running on 3000. 6 is no. of clients allowed to make the request called backlog and queue length.

System.***out***.println("Server Started");

}

**public** **void** startChart() **throws** Exception {

Socket sSocket = ssSocket.accept();

System.***out***.println("Requst Received."); // This will be print the data once data is recieved from client.

OutputStream out = sSocket.getOutputStream(); // returns an output stream for the given socket. If you close the returned OutputStream then it will close the linked socket.

   //getOutputStream() returns an output stream for writing bytes to this socket.

InputStream in = sSocket.getInputStream();

DataOutputStream dout = **new** DataOutputStream(out);

DataInputStream din = **new** DataInputStream(in);

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

**while**(**true**) {

String data = din.readUTF();

System.***out***.println(data);

data = scr.nextLine();

dout.writeUTF(data);

}

}

}

**package** networking;

**public** **class** ClientMain {

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

**try** {

Client clientApp = **new** Client();

clientApp.startChat();

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

**JDBC : Load driver. For this we need to build the jar file using Build path.**

**package** jdbc;

**public** **class** DBUtil {

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

String driverName = "oracle.jdbc.OracleDriver";

**try** {

Class.*forName*(driverName);

System.***out***.println("Driver loaded.");

} **catch** (ClassNotFoundException e) {

e.printStackTrace();

}

}

}

**For connection established:**

**package** jdbc;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**public** **class** DBUtil {

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

String driverName = "oracle.jdbc.OracleDriver";

**try** {

Class.*forName*(driverName);

System.***out***.println("Driver loaded.");

String URL = "jdbc:oracle:thin:@localhost:1521:xe";

//”jdbc:derby:C:\\users\\adm\\testdb”;

String ID = "system";

String Password = "12345";

Connection con = DriverManager.*getConnection*(URL, ID, Password);

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

**Database connection and Retrieval**

**package jdbc;**

**import java.sql.Connection;**

**import java.sql.DriverManager;**

**public class DBUtils {**

**public static Connection buildConnection() throws Exception{**

**String driverName = "oracle.jdbc.OracleDriver";**

**//Class.forName(driverName);**

**String URL = "jdbc:oracle:thin:@localhost:1521:xe";**

**String ID = "system";**

**String Password = "12345";**

**Connection con = DriverManager.getConnection(URL, ID, Password);**

**return con;**

**}**

**}**

**package** jdbc;

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

**private** **int** empId;

**private** String empName;

**private** **float** sal;

**public** Employee() {

}

**public** Employee(**int** empId, String empName, **float** sal) {

**super**();

**this**.empId = empId;

**this**.empName = empName;

**this**.sal = sal;

}

**public** **int** getEmpId() {

**return** empId;

}

**public** **void** setEmpId(**int** empId) {

**this**.empId = empId;

}

**public** String getEmpName() {

**return** empName;

}

**public** **void** setEmpName(String empName) {

**this**.empName = empName;

}

**public** **float** getSal() {

**return** sal;

}

**public** **void** setSal(**float** sal) {

**this**.sal = sal;

}

@Override

**public** String toString() {

**return** "Employee [empId=" + empId + ", empName=" + empName + ", sal=" + sal + "]";

}

@Override

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

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

**int** result = 1;

result = prime \* result + empId;

result = prime \* result + ((empName == **null**) ? 0 : empName.hashCode());

result = prime \* result + Float.*floatToIntBits*(sal);

**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** (empId != other.empId)

**return** **false**;

**if** (empName == **null**) {

**if** (other.empName != **null**)

**return** **false**;

} **else** **if** (!empName.equals(other.empName))

**return** **false**;

**if** (Float.*floatToIntBits*(sal) != Float.*floatToIntBits*(other.sal))

**return** **false**;

**return** **true**;

}

}

**package** jdbc;

**import** java.util.Collection;

**public** **interface** DaoInterface<T> {

Collection<T> getAll();

}

**package** jdbc;

**import** java.sql.Connection;

**import** java.sql.ResultSet;

**import** java.sql.Statement;

**import** java.util.ArrayList;

**import** java.util.Collection;

**public** **class** EmployeeDAO **implements** DaoInterface<Employee> {

@Override

**public** Collection<Employee> getAll() {

// Declare collection outside try block so that it will be accessible everywhere.

Collection<Employee> employees = **new** ArrayList<>();

**try**(Connection conn = DBUtils.*buildConnection*()){

Statement stmt = conn.createStatement();

String sqlquery = "select \* from emps";

ResultSet result = stmt.executeQuery(sqlquery);

**while** (result.next()) {

**int** eno = result.getInt(1);

String enm = result.getString(2);

**float** esl = result.getFloat(3);

Employee emp = **new** Employee(eno, enm, esl);

employees.add(emp);

}

}

**catch** (Exception e) {

e.printStackTrace();

}

**return** employees;

}

}

**package** jdbc;

**import** java.util.Collection;

**public** **class** EmployeeRetrieveMain {

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

DaoInterface<Employee> idao = **new** EmployeeDAO();

Collection<Employee> emplist = idao.getAll();

**int** size = emplist.size();

**if**(size !=0) {

**for** (Employee e : emplist) {

System.***out***.println(e);

}

}

**else**

System.***out***.println("No records avail. in DB");

}

}

**Database connection and retrieve specific data :**

**package jdbc;**

**import java.sql.Connection;**

**import java.sql.DriverManager;**

**public class DBUtils {**

**public static Connection buildConnection() throws Exception{**

**String driverName = "oracle.jdbc.OracleDriver";**

**//Class.forName(driverName);**

**String URL = "jdbc:oracle:thin:@localhost:1521:xe";**

**String ID = "system";**

**String Password = "12345";**

**Connection con = DriverManager.getConnection(URL, ID, Password);**

**return con;**

**}**

**}**

**package** jdbc;

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

**private** **int** empId;

**private** String empName;

**private** **float** sal;

**public** Employee() {

}

**public** Employee(**int** empId, String empName, **float** sal) {

**super**();

**this**.empId = empId;

**this**.empName = empName;

**this**.sal = sal;

}

**public** **int** getEmpId() {

**return** empId;

}

**public** **void** setEmpId(**int** empId) {

**this**.empId = empId;

}

**public** String getEmpName() {

**return** empName;

}

**public** **void** setEmpName(String empName) {

**this**.empName = empName;

}

**public** **float** getSal() {

**return** sal;

}

**public** **void** setSal(**float** sal) {

**this**.sal = sal;

}

@Override

**public** String toString() {

**return** "Employee [empId=" + empId + ", empName=" + empName + ", sal=" + sal + "]";

}

@Override

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

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

**int** result = 1;

result = prime \* result + empId;

result = prime \* result + ((empName == **null**) ? 0 : empName.hashCode());

result = prime \* result + Float.*floatToIntBits*(sal);

**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** (empId != other.empId)

**return** **false**;

**if** (empName == **null**) {

**if** (other.empName != **null**)

**return** **false**;

} **else** **if** (!empName.equals(other.empName))

**return** **false**;

**if** (Float.*floatToIntBits*(sal) != Float.*floatToIntBits*(other.sal))

**return** **false**;

**return** **true**;

}

}

**package** jdbc;

**import** java.util.Collection;

**public** **interface** DaoInterface<T, K> {

Collection<T> getAll();

T getByIdentity(K key);

}

**package** jdbc;

**import** java.sql.Connection;

**import** java.sql.PreparedStatement;

**import** java.sql.ResultSet;

**import** java.sql.Statement;

**import** java.util.ArrayList;

**import** java.util.Collection;

**public** **class** EmployeeDAO **implements** DaoInterface<Employee, Integer> {

@Override

**public** Collection<Employee> getAll() {

// Declare collection outside try block so that it will be accessible everywhere.

Collection<Employee> employees = **new** ArrayList<>();

**try**(Connection conn = DBUtils.*buildConnection*()){

Statement stmt = conn.createStatement();

String sqlquery = "select \* from emps";

ResultSet result = stmt.executeQuery(sqlquery);

**while** (result.next()) {

**int** eno = result.getInt(1);

String enm = result.getString(2);

**float** esl = result.getFloat(3);

Employee emp = **new** Employee(eno, enm, esl);

employees.add(emp);

}

}

**catch** (Exception e) {

e.printStackTrace();

}

**return** employees; }

@Override

**public** Employee getByIdentity(Integer key) {

Employee emp = **null**;

**try**(Connection conn = DBUtils.*buildConnection*()){

String sqlquery = "select \* from emps where empno = ?";

PreparedStatement prstmt = conn.prepareStatement(sqlquery);

prstmt.setInt(1, key);

ResultSet result = prstmt.executeQuery();

**if**(result.next()) {

**int** eno = result.getInt(1);

String enm = result.getString(2);

**float** esl = result.getFloat(3);

emp = **new** Employee(eno, enm, esl);

}

}

**catch**(Exception e) {

e.printStackTrace();

}

**return** emp;

}

}

**package** jdbc;

**public** **class** EmployeeRetrievebyIdMain {

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

DaoInterface<Employee, Integer> idao = **new** EmployeeDAO();

Employee emp = idao.getByIdentity(1003);

**if** (emp != **null**) {

System.***out***.println(emp);

}

**else** {

System.***out***.println("Employee not found.");

}

}

}

**Insert data in DB : DBUtil and Employee class is same , so not copied.**

**package** jdbc;

**import** java.util.Collection;

**public** **interface** DaoInterface<T, K> {

Collection<T> getAll();

T getByIdentity(K key);

**void** insert(T t);

}

**package** jdbc;

**import** java.sql.Connection;

**import** java.sql.PreparedStatement;

**import** java.sql.ResultSet;

**import** java.sql.Statement;

**import** java.util.ArrayList;

**import** java.util.Collection;

**public** **class** EmployeeDAO **implements** DaoInterface<Employee, Integer> {

@Override

**public** Collection<Employee> getAll() {

// Declare collection outside try block so that it will be accessible everywhere.

Collection<Employee> employees = **new** ArrayList<>();

**try**(Connection conn = DBUtils.*buildConnection*()){

Statement stmt = conn.createStatement();

String sqlquery = "select \* from emps";

ResultSet result = stmt.executeQuery(sqlquery);

**while** (result.next()) {

**int** eno = result.getInt(1);

String enm = result.getString(2);

**float** esl = result.getFloat(3);

Employee emp = **new** Employee(eno, enm, esl);

employees.add(emp);

}

}

**catch** (Exception e) {

e.printStackTrace();

}

**return** employees; }

@Override

**public** Employee getByIdentity(Integer key) {

Employee emp = **null**;

**try**(Connection conn = DBUtils.*buildConnection*()){

String sqlquery = "select \* from emps where empno = ?";

PreparedStatement prstmt = conn.prepareStatement(sqlquery);

prstmt.setInt(1, key);

ResultSet result = prstmt.executeQuery();

**if**(result.next()) {

**int** eno = result.getInt(1);

String enm = result.getString(2);

**float** esl = result.getFloat(3);

emp = **new** Employee(eno, enm, esl);

}

}

**catch**(Exception e) {

e.printStackTrace();

}

**return** emp;

}

@Override

**public** **void** insert(Employee emp) {

**try**(Connection conn = DBUtils.*buildConnection*()){

String sqlquery = "Insert into emps values(?, ? , ?)";

PreparedStatement prstmt = conn.prepareStatement(sqlquery);

**int** eno = emp.getEmpId();

String enm = emp.getEmpName();

**float** esl = emp.getSal();

prstmt.setInt(1, eno);

prstmt.setString(2, enm);

prstmt.setFloat(3, esl);

**int** count = prstmt.executeUpdate();

System.***out***.println(count + "Record Inserted.");

}

**catch**(Exception e) {

e.printStackTrace();

}

}

}

**package** jdbc;

**public** **class** EmployeeInsertMain {

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

DaoInterface<Employee, Integer> idao = **new** EmployeeDAO();

Employee e1 = **new** Employee(1006,"Rakesh", 5646.66f);

idao.insert(e1);

}

}

**Delete record: : DBUtil and Employee class is same , so not copied.**

**package** jdbc;

**import** java.util.Collection;

**public** **interface** DaoInterface<T, K> {

Collection<T> getAll();

T getByIdentity(K key);

**void** insert(T t);

**void** DeleteByIdentity(K key);

}

**package** jdbc;

**import** java.sql.Connection;

**import** java.sql.PreparedStatement;

**import** java.sql.ResultSet;

**import** java.sql.Statement;

**import** java.util.ArrayList;

**import** java.util.Collection;

**public** **class** EmployeeDAO **implements** DaoInterface<Employee, Integer> {

@Override

**public** Collection<Employee> getAll() {

// Declare collection outside try block so that it will be accessible everywhere.

Collection<Employee> employees = **new** ArrayList<>();

**try**(Connection conn = DBUtils.*buildConnection*()){

Statement stmt = conn.createStatement();

String sqlquery = "select \* from emps";

ResultSet result = stmt.executeQuery(sqlquery);

**while** (result.next()) {

**int** eno = result.getInt(1);

String enm = result.getString(2);

**float** esl = result.getFloat(3);

Employee emp = **new** Employee(eno, enm, esl);

employees.add(emp);

}

}

**catch** (Exception e) {

e.printStackTrace();

}

**return** employees; }

@Override

**public** Employee getByIdentity(Integer key) {

Employee emp = **null**;

**try**(Connection conn = DBUtils.*buildConnection*()){

String sqlquery = "select \* from emps where empno = ?";

PreparedStatement prstmt = conn.prepareStatement(sqlquery);

prstmt.setInt(1, key);

ResultSet result = prstmt.executeQuery();

**if**(result.next()) {

**int** eno = result.getInt(1);

String enm = result.getString(2);

**float** esl = result.getFloat(3);

emp = **new** Employee(eno, enm, esl);

}

}

**catch**(Exception e) {

e.printStackTrace();

}

**return** emp;

}

@Override

**public** **void** insert(Employee emp) {

**try**(Connection conn = DBUtils.*buildConnection*()){

String sqlquery = "Insert into emps values(?, ? , ?)";

PreparedStatement prstmt = conn.prepareStatement(sqlquery);

**int** eno = emp.getEmpId();

String enm = emp.getEmpName();

**float** esl = emp.getSal();

prstmt.setInt(1, eno);

prstmt.setString(2, enm);

prstmt.setFloat(3, esl);

**int** count = prstmt.executeUpdate();

System.***out***.println(count + "Record Inserted.");

}

**catch**(Exception e) {

e.printStackTrace();

}

}

@Override

**public** **void** DeleteByIdentity(Integer key) {

**try**(Connection conn = DBUtils.*buildConnection*()){

String sqlquery = "Delete From emps where empno = ?";

PreparedStatement prstmt = conn.prepareStatement(sqlquery);

prstmt.setInt(1, key);

**int** count = prstmt.executeUpdate();

System.***out***.println(count + "Record deleted.");

}

**catch**(Exception e) {

e.printStackTrace();

}

}

}

**package** jdbc;

**public** **class** EmployeeDeleteMain {

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

DaoInterface<Employee, Integer> idao = **new** EmployeeDAO();

idao.DeleteByIdentity(1006);

}

}

**Update Record : copied only daoInterface , Main method**

**package jdbc;**

**import java.sql.Connection;**

**import java.sql.PreparedStatement;**

**import java.sql.ResultSet;**

**import java.sql.Statement;**

**import java.util.ArrayList;**

**import java.util.Collection;**

**public class EmployeeDAO implements DaoInterface<Employee, Integer> {**

**@Override**

**public Collection<Employee> getAll() {**

**// Declare collection outside try block so that it will be accessible everywhere.**

**Collection<Employee> employees = new ArrayList<>();**

**try(Connection conn = DBUtils.buildConnection()){**

**Statement stmt = conn.createStatement();**

**String sqlquery = "select \* from emps";**

**ResultSet result = stmt.executeQuery(sqlquery);**

**while (result.next()) {**

**int eno = result.getInt(1);**

**String enm = result.getString(2);**

**float esl = result.getFloat(3);**

**Employee emp = new Employee(eno, enm, esl);**

**employees.add(emp);**

**}**

**}**

**catch (Exception e) {**

**e.printStackTrace();**

**}**

**return employees; }**

**@Override**

**public Employee getByIdentity(Integer key) {**

**Employee emp = null;**

**try(Connection conn = DBUtils.buildConnection()){**

**String sqlquery = "select \* from emps where empno = ?";**

**PreparedStatement prstmt = conn.prepareStatement(sqlquery);**

**prstmt.setInt(1, key);**

**ResultSet result = prstmt.executeQuery();**

**if(result.next()) {**

**int eno = result.getInt(1);**

**String enm = result.getString(2);**

**float esl = result.getFloat(3);**

**emp = new Employee(eno, enm, esl);**

**}**

**}**

**catch(Exception e) {**

**e.printStackTrace();**

**}**

**return emp;**

**}**

**@Override**

**public void insert(Employee emp) {**

**try(Connection conn = DBUtils.buildConnection()){**

**String sqlquery = "Insert into emps values(?, ? , ?)";**

**PreparedStatement prstmt = conn.prepareStatement(sqlquery);**

**int eno = emp.getEmpId();**

**String enm = emp.getEmpName();**

**float esl = emp.getSal();**

**prstmt.setInt(1, eno);**

**prstmt.setString(2, enm);**

**prstmt.setFloat(3, esl);**

**int count = prstmt.executeUpdate();**

**System.out.println(count + "Record Inserted.");**

**}**

**catch(Exception e) {**

**e.printStackTrace();**

**}**

**}**

**@Override**

**public void DeleteByIdentity(Integer key) {**

**try(Connection conn = DBUtils.buildConnection()){**

**String sqlquery = "Delete From emps where empno = ?";**

**PreparedStatement prstmt = conn.prepareStatement(sqlquery);**

**prstmt.setInt(1, key);**

**int count = prstmt.executeUpdate();**

**System.out.println(count + "Record deleted.");**

**}**

**catch(Exception e) {**

**e.printStackTrace();**

**}**

**}**

**@Override**

**public void update(Employee t) {**

**try(Connection conn = DBUtils.buildConnection()){**

**String sqlquery = "update emps set ename = ? , sal = ? where empno = ? ";**

**PreparedStatement prstmt = conn.prepareStatement(sqlquery);**

**int eno = t.getEmpId();**

**String enm = t.getEmpName();**

**float esl = t.getSal();**

**prstmt.setString(1, enm);**

**prstmt.setFloat(2, esl);**

**prstmt.setInt(3, eno);**

**int count = prstmt.executeUpdate();**

**System.out.println(count + "Record updated.");**

**}**

**catch(Exception e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**package** jdbc;

**public** **class** EmployeeUpdateMain {

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

DaoInterface<Employee, Integer> idao = **new** EmployeeDAO();

Employee emp = idao.getByIdentity(1003);

**if**(emp != **null**) {

emp.setEmpName("Dipti");

emp.setSal(5646.66f);

idao.update(emp);

}

**else** {

System.***out***.println("Record not updated.");

}

}

}

**Callable Statement : save below file as SQL file name like plsql.sql**

**create or replace procedure**

**raise\_salary( eno in number, amt in number)**

**as**

**begin**

**update emps set sal= sal+ amt**

**where empno = eno;**

**end;**

**/**

**package** jdbc;

**import** java.sql.CallableStatement;

**import** java.sql.Connection;

**import** java.sql.SQLException;

**public** **class** ProcedureInvocationMain {

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

**try**(Connection con = DBUtils.*buildConnection*()){

String queryString = "{call raise\_salary(?,?)}";

CallableStatement cstmt = con.prepareCall(queryString);

cstmt.setInt(1, 1004);

cstmt.setInt(2, 10000);

cstmt.execute();

System.***out***.println("Procedure Invoked");

}

**catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

**Procedure with in and out parameters: out or print salary**

**create or replace procedure**

**get\_salary( eno in number, salary out number)**

**as**

**begin**

**select sal into salary from emps**

**where empno = eno;**

**end;**

**/**

**Through sql command prompt : follow below steps 1,2,3**

1. **SQL> @C:\Users\adm\Documents\DJ57741\plsql2**

**Procedure created.**

1. **SQL> variable sl number;**
2. **SQL> exec get\_salary(1003, :sl);**

**PL/SQL procedure successfully completed.**

**SQL> print sl**

**SL**

**----------**

**5646.66**

**Java code for the same above sql query : to execute procedure.**

**package** jdbc;

**import** java.sql.CallableStatement;

**import** java.sql.Connection;

**import** java.sql.Types;

**public** **class** ProcedureOutInvocationMain {

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

**try**(Connection con = DBUtils.*buildConnection*()){

String queryString = "{call get\_salary(?,?)}";

CallableStatement cstmt = con.prepareCall(queryString);

cstmt.setInt(1, 1004);

cstmt.registerOutParameter(2, Types.***FLOAT***);

cstmt.execute();

System.***out***.println("Procedure Invoked");

**float** sal = cstmt.getFloat(2);

System.***out***.println(sal);

}

**catch**(Exception e) {

e.printStackTrace();

}

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*JAVA Core Ends here \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*