1)//Hierarchal inheritance method overriding dynamic method dispatch

class Animal3 {

// Method to be overridden

void sound() {

System.out.println("Animal makes a sound");

}

}

// Child class 1 (Dog extends Animal)

class Dog4 extends Animal3 {

// Overriding the sound method in Dog

@Override

void sound() {

System.out.println("Dog barks");

}

}

// Child class 2 (Cat extends Animal)

class Cat5 extends Animal3 {

// Overriding the sound method in Cat

@Override

void sound() {

System.out.println("Cat meows");

}

}

// Main class to demonstrate dynamic method dispatch

public class HirarInheritancedemo {

public static void main(String[] args) {

// Creating objects of Dog and Cat

Animal3 animal1 = new Dog4(); // Animal reference, Dog object

Animal3 animal2 = new Cat5(); // Animal reference, Cat object

// Demonstrating dynamic method dispatch

animal1.sound(); // Calls Dog's sound() method

animal2.sound(); // Calls Cat's sound() method

}

}

2)//Write a java program to implement looping and jumping statements.

public class LoopingAndJumpingDemo {

public static void main(String[] args) {

// For loop example

System.out.println("For Loop:");

for (int i = 1; i <= 5; i++) {

if (i == 3) {

continue; // Skip when i is 3

}

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

}

// While loop example

System.out.println("\nWhile Loop:");

int j = 1;

while (j <= 5) {

if (j == 4) {

break; // Stop the loop when j is 4

}

System.out.println("j = " + j);

j++;

}

// Do-while loop example

System.out.println("\nDo-While Loop:");

int k = 1;

do {

System.out.println("k = " + k);

k++;

} while (k <= 5);

}

}

3)//Write a java program for creating a runnable threads

class MyRunnable implements Runnable {

// Override the run() method to specify what the thread will do

@Override

public void run() {

for (int i = 1; i <= 5; i++) {

// Print the current thread's name and count value

System.out.println(Thread.currentThread().getName() + " - Count: " + i);

// Pause the thread for 500 milliseconds

try {

Thread.sleep(500);

} catch (InterruptedException e) {

System.out.println(Thread.currentThread().getName() + " was interrupted.");

}

}

System.out.println(Thread.currentThread().getName() + " finished.");

}

}

// Main class to run the program

public class RunnableExample {

public static void main(String[] args) {

// Create an instance of MyRunnable

MyRunnable runnableTask = new MyRunnable();

// Create two Thread objects, passing the same runnableTask instance

Thread thread1 = new Thread(runnableTask, "Thread 1");

Thread thread2 = new Thread(runnableTask, "Thread 2");

// Start both threads

thread1.start();

thread2.start();

}

}

4)//Write a java program for creating abstract class

abstract class NIE {

// Abstract method (no body)

abstract void display();

}

// Abstract class student extending NIE

abstract class Student extends NIE {

// Concrete method with implementation

@Override

void display() {

System.out.println("NIE college");

}

// Abstract method for student activities

abstract void study();

}

// Concrete class implementing Student

class EngineeringStudent extends Student {

@Override

void study() {

System.out.println("Students are studying.");

}

}

// Main class

public class demo1 {

public static void main(String[] args) {

// Create an instance of EngineeringStudent

EngineeringStudent obj = new EngineeringStudent();

obj.display(); // Calls display method

obj.study(); // Calls study method

}

}

5)//Write a java program for implement interfaces class and method

interface Vehicle {

// Abstract method (to be implemented by the class)

void start();

// Static method in the interface

static void showType() {

System.out.println("This is a Vehicle");

}

}

// Abstract class declaration

abstract class Engine {

// Abstract method (to be implemented by the class)

abstract void fuelType();

// Concrete (regular) method

void engineInfo() {

System.out.println("This engine is standard");

}

}

// Class that extends the abstract class and implements the interface

class Car extends Engine implements Vehicle {

// Implementing the start() method from the Vehicle interface

@Override

public void start() {

System.out.println("Car is starting");

}

// Implementing the fuelType() method from the Engine abstract class

@Override

void fuelType() {

System.out.println("Car uses petrol");

}

}

// Main class

public class inter {

public static void main(String[] args) {

// Calling the static method from the Vehicle interface

Vehicle.showType();

// Creating an object of the Car class

Car myCar = new Car();

// Calling methods implemented in the Car class

myCar.start(); // Calls the start() method

myCar.fuelType(); // Calls the fuelType() method

myCar.engineInfo(); // Calls the engineInfo() method

System.out.println("Program finished");

}

}

6)//Write a java method overriding and constructor overloading

class Animal {

// Constructor 1 (no parameters)

Animal() {

System.out.println("An animal is Born");

}

// Constructor 2 (with one parameter)

Animal(String name) {

System.out.println("Animal name is: " + name);

}

// Method to be overridden

void sound() {

System.out.println("Animal makes a sound");

}

}

// Subclass

class Dog extends Animal {

// Constructor that calls the superclass constructor

Dog() {

super("Dog"); // Calling superclass constructor with one parameter

System.out.println("Dog is Born");

}

// Method overriding

@Override

void sound() {

System.out.println("Dog barks");

}

}

// Main class

public class Main1 {

public static void main(String[] args) {

// Constructor overloading demonstration

Animal animal1 = new Animal(); // Calls no-argument constructor

Animal animal2 = new Animal("Cat"); // Calls one-argument constructor

// Method overriding demonstration

Dog dog = new Dog(); // Calls Dog constructor

dog.sound(); // Calls overridden method in Dog class

}

}

7)//Write a java program for control statements and looping statements

public class ControlAndLoopingDemo {

public static void main(String[] args) {

// 1. if-else statement

int num = 10;

System.out.println("Using if-else statement:");

if (num > 0) {

System.out.println("The number is positive.");

} else {

System.out.println("The number is non-positive.");

}

// 2. switch statement

int day = 3;

System.out.println("\nUsing switch statement:");

switch (day) {

case 1:

System.out.println("Sunday");

break;

case 2:

System.out.println("Monday");

break;

case 3:

System.out.println("Tuesday");

break;

default:

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

}

// 3. for loop

System.out.println("\nUsing for loop:");

for (int i = 1; i <= 5; i++) {

System.out.println("Iteration: " + i);

}

// 4. while loop

System.out.println("\nUsing while loop:");

int count = 1;

while (count <= 3) {

System.out.println("Count: " + count);

count++;

}

// 5. do-while loop

System.out.println("\nUsing do-while loop:");

int num2 = 1;

do {

System.out.println("Num: " + num2);

num2++;

} while (num2 <= 3);

}

}

8)//Write a java program for enum creation of constructor enum methods

enum week

{

mon(1),tue(2),wed(3),thrus(4),fri(5),sat(6),sun(7);

int num;

week(int n)

{

num=n;

}

void displayweek()

{

switch(num)

{

case 1:System.out.println("1 day of the week");

break;

case 2:System.out.println("2 day of the week");

break;

case 3:System.out.println("3 day of the week");

break;

case 4:System.out.println("4 day of the week");

break;

case 5:System.out.println("5 day of the week");

break;

case 6:System.out.println("6 day of the week");

break;

case 7:System.out.println("7 day of the week");

break;

}

}

}

class enumclass{

public static void main(String args[])

{

System.out.println("weeks of the day");

week w[]=week.values();

for(week wk:w)

{

System.out.println(wk);

wk.displayweek();

System.out.println();

}

week bd=week.valueOf("wed");

bd.displayweek();

}

}

9)//Write a java program for creating a extending thread class

class MyRunnable implements Runnable {

// Override the run() method to specify what the thread will do

@Override

public void run() {

for (int i = 1; i <= 5; i++) {

// Print the current thread's name and count value

System.out.println(Thread.currentThread().getName() + " - Count: " + i);

// Pause the thread for 500 milliseconds

try {

Thread.sleep(500);

} catch (InterruptedException e) {

System.out.println(Thread.currentThread().getName() + " was interrupted.");

}

}

System.out.println(Thread.currentThread().getName() + " finished.");

}

}

// Main class to run the program

public class ExtendingThread {

public static void main(String[] args) {

// Create an instance of MyRunnable

MyRunnable runnableTask = new MyRunnable();

// Create two Thread objects, passing the same runnableTask instance

Thread thread1 = new Thread(runnableTask, "Thread 1");

Thread thread2 = new Thread(runnableTask, "Thread 2");

// Start both threads

thread1.start();

thread2.start();

}

}

10)//Implement all types of inheritance in one program. Call the constructor using super key word

class Animal2 {

// Constructor of Animal

Animal2() {

System.out.println("Animal is created");

}

void eat() {

System.out.println("Animal eats");

}

}

// Derived class 1 (Single Inheritance)

class Dog3 extends Animal2 {

// Constructor of Dog

Dog3() {

super(); // Calls the constructor of Animal

System.out.println("Dog is created");

}

void bark() {

System.out.println("Dog barks");

}

}

// Derived class 2 (Multilevel Inheritance)

class Puppy extends Dog3 {

// Constructor of Puppy

Puppy() {

super(); // Calls the constructor of Dog

System.out.println("Puppy is created");

}

void weep() {

System.out.println("Puppy weeps");

}

}

// Another derived class (Hierarchical Inheritance)

class Cat4 extends Animal2 {

// Constructor of Cat

Cat4() {

super(); // Calls the constructor of Animal

System.out.println("Cat is created");

}

void meow() {

System.out.println("Cat meows");

}

}

// Main class

public class InheritanceDemo {

public static void main(String[] args) {

// Single Inheritance: Creating Dog object

System.out.println("Creating Dog object:");

Dog3 dog3 = new Dog3();

dog3.eat(); // From Animal

dog3.bark(); // From Dog

System.out.println();

// Multilevel Inheritance: Creating Puppy object

System.out.println("Creating Puppy object:");

Puppy puppy = new Puppy();

puppy.eat(); // From Animal

puppy.bark(); // From Dog

puppy.weep(); // From Puppy

System.out.println();

// Hierarchical Inheritance: Creating Cat object

System.out.println("Creating Cat object:");

Cat4 cat4 = new Cat4();

cat4.eat(); // From Animal

cat4.meow(); // From Cat

}

}

11)//exception using Nested try,multiple catch,finally,throw

class MyExceptionExample extends Exception {

public MyExceptionExample(String str) {

super(str); // Use super to call the Exception constructor

}

}

public class ExceptionDemo {

// Method to validate age

static void validate(int age) throws Exception {

if (age < 18) {

// Throw custom exception if age is less than 18

throw new Exception("Age is not valid");

} else {

System.out.println("Age is valid");

}

}

public static void main(String[] args) {

try {

System.out.println("Checking if age is valid or not:");

// First inner try-catch block

try {

validate(13); // Invalid age, should throw exception

} catch (Exception e) {

System.out.println("Exception caught: " + e.getMessage());

}

// Second inner try-catch block

try {

validate(20); // Valid age, no exception should be thrown

} catch (Exception e) {

System.out.println("Eligible to vote: " + e.getMessage());

}

} finally {

// Finally block always executes

System.out.println("Exiting the program");

}

System.out.println("Program ends normally");

}

}

12)//custom Exception implementing own

class MyExceptionExample extends Exception {

public MyExceptionExample(String str) {

super(str); // Use super to call the Exception constructor

}

}

public class ExceptionDemo {

// Method to validate age

static void validate(int age) throws Exception {

if (age < 18) {

// Throw custom exception if age is less than 18

throw new Exception("Age is not valid");

} else {

System.out.println("Age is valid");

}

}

public static void main(String[] args) {

// First inner try-catch block

try {

validate(13); // Invalid age, should throw exception

} catch (Exception e) {

System.out.println("Exception caught: " + e.getMessage());

} finally {

// Finally block always executes

System.out.println("Exiting the program");

}

}

}