Constructors and methods with inheritance (Single and Multilevel Inheritance)

1. Write java code for creating 3 methods called even, odd and prime in parent class and override them from child class. The parent class should print 1 to 20 range in all method, but child print 1 to 30 array range in all methods. And access all the methods using derived class object

**CODE:**

class Parent {

void even() {

System.out.println("Even numbers from 1 to 20:");

for (int i = 1; i <= 20; i++) {

if (i % 2 == 0) {

System.out.println(i);

}

}

}

void odd() {

System.out.println("Odd numbers from 1 to 20:");

for (int i = 1; i <= 20; i++) {

if (i % 2 != 0) {

System.out.println(i);

}

}

}

void prime() {

System.out.println("Prime numbers from 1 to 20:");

for (int i = 1; i <= 20; i++) {

if (isPrime(i)) {

System.out.println(i);

}

}

}

boolean isPrime(int n) {

if (n <= 1) {

return false;

}

for (int i = 2; i < n; i++) {

if (n % i == 0) {

return false;

}

}

return true;

}

}

class Child extends Parent {

@Override

void even() {

System.out.println("Even numbers from 1 to 30:");

for (int i = 1; i <= 30; i++) {

if (i % 2 == 0) {

System.out.println(i);

}

}

}

@Override

void odd() {

System.out.println("Odd numbers from 1 to 30:");

for (int i = 1; i <= 30; i++) {

if (i % 2 != 0) {

System.out.println(i);

}

}

}

@Override

void prime() {

System.out.println("Prime numbers from 1 to 30:");

for (int i = 1; i <= 30; i++) {

if (isPrime(i)) {

System.out.println(i);

}

}

}

}

public class Main {

public static void main(String[] args) {

Child child = new Child();

child.even();

System.out.println();

child.odd();

System.out.println();

child.prime();

}

}

**OUTPUT:**  
Even numbers from 1 to 30:

2

4

6

8

10

12

14

16

18

20

22

24

26

28

30

Odd numbers from 1 to 30:

1

3

5

7

9

11

13

15

17

19

21

23

25

27

29

Prime numbers from 1 to 30:

2

3

5

7

11

13

17

19

23

29

1. Override a default constructor, which contains a=10,b=15 in java using parametrized constructor which contains a=20,b=25. Create a method called disp(). And call both the constructors associating it with disp()

**CODE:**  
class MyClass {

int a;

int b;

// Default constructor

MyClass() {

this.a = 10;

this.b = 15;

}

// Parameterized constructor

MyClass(int a, int b) {

this.a = a;

this.b = b;

}

void disp() {

System.out.println("a = " + a + ", b = " + b);

}

}

public class Main {

public static void main(String[] args) {

// Create an object using the default constructor

MyClass obj1 = new MyClass();

obj1.disp();

// Create an object using the parameterized constructor

MyClass obj2 = new MyClass(20, 25);

obj2.disp();

}

}

**OUTPUT:**  
a = 10, b = 15

a = 20, b = 25

3) Java program to create a Base class with a method called ‘void area(int a, int b)’ which print are a of a square. Now override the method from Derived class and make it print area of a rectangle.

**CODE:**

class Base {

void area(int a, int b) {

int squareArea = a \* a;

System.out.println("Area of the square: " + squareArea);

}

}

class Derived extends Base {

@Override

void area(int a, int b) {

int rectangleArea = a \* b;

System.out.println("Area of the rectangle: " + rectangleArea);

}

}

public class Main {

public static void main(String[] args) {

Base base = new Base();

base.area(4, 5); // This will call the method in the Base class

Derived derived = new Derived();

derived.area(4, 5); // This will call the overridden method in the Derived class

}

}

**OUTPUT:**  
Area of the square: 16

Area of the rectangle: 20

4) Write java code to overload a method called ‘int sum(int a, int b)’ by the 3 ways.

a) By increasing and decreasing no. of parameters

**CODE:**

public class Main {

// Original method with two parameters

int sum(int a, int b) {

return a + b;

}

// Overloaded method with three parameters

int sum(int a, int b, int c) {

return a + b + c;

}

// Overloaded method with one parameter

int sum(int a) {

return a;

}

public static void main(String[] args) {

Main main = new Main();

System.out.println("Sum of 2 and 3: " + main.sum(2, 3)); // This will call the original method

System.out.println("Sum of 2, 3 and 4: " + main.sum(2, 3, 4)); // This will call the overloaded method with three parameters

System.out.println("Sum of 2: " + main.sum(2)); // This will call the overloaded method with one parameter

}

}

**OUTPUT:**  
Sum of 2 and 3: 5

Sum of 2, 3 and 4: 9

Sum of 2: 2

b) By changing the data types of parameters

**CODE:**  
public class Main {

// Original method with int parameters

int sum(int a, int b) {

return a + b;

}

// Overloaded method with double parameters

double sum(double a, double b) {

return a + b;

}

public static void main(String[] args) {

Main main = new Main();

System.out.println("Sum of 2 and 3: " + main.sum(2, 3)); // This will call the original method

System.out.println("Sum of 2.5 and 3.5: " + main.sum(2.5, 3.5)); // This will call the overloaded method with double parameters

}

}

**OUTPUT:**  
Sum of 2 and 3: 5

Sum of 2.5 and 3.5: 6.0

c) By interchanging the parameters

**CODE:**

public class Main {

// Original method

public int sum(int a, int b) {

return a + b;

}

// Overloaded method with different types of parameters

public int sum(double a, double b) {

return (int)(a + b);

}

public static void main(String[] args) {

Main main = new Main();

// Using the first sum method

int sum1 = main.sum(1, 2);

System.out.println("Sum of two integers: " + sum1);

// Using the overloaded sum method

int sum2 = main.sum(1.5, 2.5);

System.out.println("Sum of two doubles: " + sum2);

}

}

**OUTPUT:**  
Sum of two integers: 3

Sum of two doubles: 4

**ABSTRACT CLASSES AND INTERFACES**

5. Create an abstract class with 2 abstract methods(total() and average()) and 3 concrete methods(mean(), mode(), median()). Now extend the abstract class from a concrete class and use all the methods in that abstract class

**CODE:**

import java.util.Arrays;

// Abstract class

abstract class AbstractMath {

// Abstract methods

abstract int total(int[] numbers);

abstract double average(int[] numbers);

// Concrete methods

double mean(int[] numbers) {

return total(numbers) / (double) numbers.length;

}

int mode(int[] numbers) {

// Logic for finding mode (most frequent element) goes here

// For simplicity, let's assume the first element is the mode

return numbers[0];

}

double median(int[] numbers) {

// Logic for finding median (middle element) goes here

// For simplicity, let's sort the array and find the middle element

Arrays.sort(numbers);

int n = numbers.length;

if (n % 2 == 0) {

return (numbers[n/2 - 1] + numbers[n/2]) / 2.0;

} else {

return numbers[n/2];

}

}

}

// Concrete class extending AbstractMath

class MathOperations extends AbstractMath {

// Implementing abstract methods

int total(int[] numbers) {

int sum = 0;

for (int num : numbers) {

sum += num;

}

return sum;

}

double average(int[] numbers) {

return total(numbers) / (double) numbers.length;

}

}

public class Java {

public static void main(String[] args) {

// Example usage

int[] numbers = {3, 7, 2, 5, 8};

// Create an instance of the concrete class

MathOperations math = new MathOperations();

// Use all methods

System.out.println("Total: " + math.total(numbers));

System.out.println("Average: " + math.average(numbers));

System.out.println("Mean: " + math.mean(numbers));

System.out.println("Mode: " + math.mode(numbers));

System.out.println("Median: " + math.median(numbers));

}

}

**OUTPUT:**

Total: 25

Average: 5.0

Mean: 5.0

Mode: 3

Median: 5.0

6. Create an interface with 4 methods called add(), sub(), mul() and div(). Then give implementation for all in the implementing class

**CODE:**

// Interface

interface Calculator {

int add(int a, int b);

int sub(int a, int b);

int mul(int a, int b);

double div(int a, int b);

}

// Implementing class

class BasicCalculator implements Calculator {

@Override

public int add(int a, int b) {

return a + b;

}

@Override

public int sub(int a, int b) {

return a - b;

}

@Override

public int mul(int a, int b) {

return a \* b;

}

@Override

public double div(int a, int b) {

if (b == 0) {

throw new ArithmeticException("Cannot divide by zero");

}

return (double) a / b;

}

}

public class Java {

public static void main(String[] args) {

// Create an instance of the implementing class

BasicCalculator calculator = new BasicCalculator();

// Use the methods

int sum = calculator.add(5, 3);

int difference = calculator.sub(10, 4);

int product = calculator.mul(6, 2);

double quotient = calculator.div(15, 3);

// Display the results

System.out.println("Sum: " + sum);

System.out.println("Difference: " + difference);

System.out.println("Product: " + product);

System.out.println("Quotient: " + quotient);

}

}

**OUTPUT:**

Sum: 8

Difference: 6

Product: 12

Quotient: 5.0

7. Create 3 interfaces with 1 method each sum(), avg(), percentage() respectively. Now implement all the 3 interfaces in your class

**CODE:**

// Interface for sum

interface Summable {

int sum(int[] numbers);

}

// Interface for average

interface Averagable {

double avg(int[] numbers);

}

// Interface for percentage

interface PercentageCalculable {

double percentage(double obtained, double total);

}

// Class implementing all three interfaces

class Calculation implements Summable, Averagable, PercentageCalculable {

@Override

public int sum(int[] numbers) {

int total = 0;

for (int num : numbers) {

total += num;

}

return total;

}

@Override

public double avg(int[] numbers) {

return (double) sum(numbers) / numbers.length;

}

@Override

public double percentage(double obtained, double total) {

return (obtained / total) \* 100.0;

}

}

public class Java {

public static void main(String[] args) {

// Example usage

int[] numbers = {80, 75, 90, 85, 95};

// Create an instance of the class implementing the interfaces

Calculation calculation = new Calculation();

// Use the methods from the implemented interfaces

int sum = calculation.sum(numbers);

double avg = calculation.avg(numbers);

double percentage = calculation.percentage(350, 500); // Obtained marks: 350, Total marks: 500

// Display the results

System.out.println("Sum: " + sum);

System.out.println("Average: " + avg);

System.out.println("Percentage: " + percentage + "%");

}

}

**OUTPUT:**

Sum: 425

Average: 85.0

Percentage: 70.0%

8. Create an interface called Tree and extend 2 classes from it called Branch1 and Branch2. Tree should contain methods fruits(), leaves() and flowers(), these methods contain 2,3,4 parameters respectively.

**CODE:**

// Interface Tree

interface Tree {

void fruits(String fruit1, String fruit2);

void leaves(String leaf1, String leaf2, String leaf3);

void flowers(String flower1, String flower2, String flower3, String flower4);

}

// Class Branch1 implementing Tree

class Branch1 implements Tree {

@Override

public void fruits(String fruit1, String fruit2) {

System.out.println("Fruits on Branch1: " + fruit1 + ", " + fruit2);

}

@Override

public void leaves(String leaf1, String leaf2, String leaf3) {

System.out.println("Leaves on Branch1: " + leaf1 + ", " + leaf2 + ", " + leaf3);

}

@Override

public void flowers(String flower1, String flower2, String flower3, String flower4) {

System.out.println("Flowers on Branch1: " + flower1 + ", " + flower2 + ", " + flower3 + ", " + flower4);

}

}

// Class Branch2 implementing Tree

class Branch2 implements Tree {

@Override

public void fruits(String fruit1, String fruit2) {

System.out.println("Fruits on Branch2: " + fruit1 + ", " + fruit2);

}

@Override

public void leaves(String leaf1, String leaf2, String leaf3) {

System.out.println("Leaves on Branch2: " + leaf1 + ", " + leaf2 + ", " + leaf3);

}

@Override

public void flowers(String flower1, String flower2, String flower3, String flower4) {

System.out.println("Flowers on Branch2: " + flower1 + ", " + flower2 + ", " + flower3 + ", " + flower4);

}

}

public class Java {

public static void main(String[] args) {

// Creating objects of Branch1 and Branch2

Branch1 branch1 = new Branch1();

Branch2 branch2 = new Branch2();

// Calling methods of Branch1

branch1.fruits("Apple", "Orange");

branch1.leaves("Green", "Yellow", "Red");

branch1.flowers("Rose", "Lily", "Daisy", "Sunflower");

// Calling methods of Branch2

branch2.fruits("Mango", "Banana");

branch2.leaves("Brown", "Orange", "Purple");

branch2.flowers("Tulip", "Daffodil", "Hibiscus", "Cherry Blossom");

}

}

**OUTPUT:**

Fruits on Branch1: Apple, Orange

Leaves on Branch1: Green, Yellow, Red

Flowers on Branch1: Rose, Lily, Daisy, Sunflower

Fruits on Branch2: Mango, Banana

Leaves on Branch2: Brown, Orange, Purple

Flowers on Branch2: Tulip, Daffodil, Hibiscus, Cherry Blossom

Final, Static, this and super keywords

9. Use static keyword in the following levels

a) Static variable

**CODE:**

class MyClass {

// This is a static variable

static int count = 0;

MyClass() {

// Increment the static variable for each new instance of MyClass

count++;

}

void displayCount() {

System.out.println("Number of instances created: " + count);

}

}

public class Main {

public static void main(String[] args) {

MyClass obj1 = new MyClass();

obj1.displayCount(); // Output: Number of instances created: 1

MyClass obj2 = new MyClass();

obj2.displayCount(); // Output: Number of instances created: 2

}

}

**OUTPUT:**

Number of instances created: 1

Number of instances created: 2

b) Static method

**CODE:**

class MyClass {

// This is a static method

static void displayMessage() {

System.out.println("Hello from the static method!");

}

}

public class Main {

public static void main(String[] args) {

// Call the static method on the class itself, not on an instance of the class

MyClass.displayMessage(); // Output: Hello from the static method!

}

}

**OUTPUT:**  
Hello from the static method!

c) Static block

**CODE:**

class MyClass {

static int num;

// This is a static block

static {

num = 10;

System.out.println("Static block executed. num: " + num);

}

}

public class Main {

public static void main(String[] args) {

// The static block will be executed when the MyClass class is loaded

MyClass obj = new MyClass();

}

}

**OUTPUT:**

Static block executed. num: 10

d) Static nested classes

**CODE:**  
class OuterClass {

static int outerNum = 10;

// This is a static nested class

static class StaticNestedClass {

void display() {

System.out.println("outerNum from OuterClass: " + outerNum);

}

}

}

public class Main {

public static void main(String[] args) {

// Create an instance of the static nested class

OuterClass.StaticNestedClass nestedObj = new OuterClass.StaticNestedClass();

nestedObj.display(); // Output: outerNum from OuterClass: 10

}

}

**OUTPUT:**  
outerNum from OuterClass: 10

10. Use Final keyword in the following levels

a) Final variable

**CODE:**

class MyClass {

// This is a final variable

final int num = 10;

void display() {

System.out.println("num: " + num);

}

}

public class Main {

public static void main(String[] args) {

MyClass obj = new MyClass();

obj.display(); // Output: num: 10

}

}

**OUTPUT:**

num: 10

b) Final method

**CODE:**

class Parent {

// This is a final method

final void display() {

System.out.println("Hello from Parent!");

}

}

class Child extends Parent {

// This method cannot override the final method in the parent class

// void display() {

// System.out.println("Hello from Child!");

// }

// New method in Child class

void sayHello() {

System.out.println("Hello from Child!");

}

}

public class Main {

public static void main(String[] args) {

Child obj = new Child();

obj.display(); // Output: Hello from Parent!

obj.sayHello(); // Output: Hello from Child!

}

}

**OUTPUT:**

Hello from Parent!

Hello from Child!

c) Final classes

**CODE:**

// This is a final class

final class FinalClass {

void display() {

System.out.println("Hello from FinalClass!");

}

}

// This class cannot extend the final class

// class Child extends FinalClass {

// // Some code...

// }

public class Main {

public static void main(String[] args) {

FinalClass obj = new FinalClass();

obj.display(); // Output: Hello from FinalClass!

}

}

**OUTPUT:**

Hello from FinalClass!

11. Use ‘this’ keyword in the following purposes

a) Referencing instance variable

**CODE:**

public class ReferencingInstanceVariable {  
 private int number;  
  
 public void setNumber(int number) {  
 // Use 'this' to refer to the instance variable  
 this.number = number;  
 }  
  
 public int getNumber() {  
 return this.number;  
 }  
  
 public static void main(String[] args) {  
 ReferencingInstanceVariable obj = new ReferencingInstanceVariable();  
 obj.setNumber(42);  
  
 System.out.println("Number: " + obj.getNumber());  
 }  
}

OUTPUT:

Number: 42

b) Invoking another constructor

**CODE:**

public class InvokingAnotherConstructor {  
 private int number;  
 private String text;  
  
 // Constructor with two parameters  
 public InvokingAnotherConstructor(int number, String text) {  
 this.number = number;  
 this.text = text;  
 }  
  
 // Constructor with one parameter, invoking the two-parameter constructor using 'this'  
 public InvokingAnotherConstructor(int number) {  
 this(number, "Default Text");  
 }  
  
 public void displayValues() {  
 System.out.println("Number: " + number);  
 System.out.println("Text: " + text);  
 }  
  
 public static void main(String[] args) {  
 InvokingAnotherConstructor obj1 = new InvokingAnotherConstructor(42, "Hello, World!");  
 obj1.displayValues();  
  
 InvokingAnotherConstructor obj2 = new InvokingAnotherConstructor(17);  
 obj2.displayValues();  
 }  
}

**OUTPUT:**

Number: 42

Text: Hello, World!

Number: 17

Text: Default Text

c) Passing current object as a parameter

**CODE:**

public class PassingCurrentObjectAsParameter {

private int number;

public PassingCurrentObjectAsParameter(int number) {

this.number = number;

}

public void processData(PassingCurrentObjectAsParameter anotherObject) {

int sum = this.number + anotherObject.number;

System.*out*.println("Sum: " + sum);

}

public static void main(String[] args) {

PassingCurrentObjectAsParameter obj1 = new PassingCurrentObjectAsParameter(30);

PassingCurrentObjectAsParameter obj2 = new PassingCurrentObjectAsParameter(15);

obj1.processData(obj2);

}

}

**OUTPUT:**  
Sum: 45

d) Returning current object

**CODE:**

public class ReturningCurrentObject {

private int number;

public ReturningCurrentObject setNumberAndReturn(int number) {

this.number = number;

return this;

}

public void displayValue() {

System.*out*.println("Number: " + number);

}

public static void main(String[] args) {

ReturningCurrentObject obj = new ReturningCurrentObject();

obj.setNumberAndReturn(10).displayValue();

}

}

**Output:**

Number:10

12. Use ‘super’ keyword in the following purposes

a) Accessing superclass members

**CODE:**

public class Animal1 {

String sound = "Animal Sound";

}

class Dog1 extends Animal1 {

void displaySound() {

// Accessing the superclass member using 'super'

System.out.println("Dog Sound: " + super.sound);

}

}

public class SuperKeywordExample {

public static void main(String[] args) {

Dog1 myDog = new Dog1();

myDog.displaySound();

}

}

**OUTPUT:**

Dog Sound :Animal Sound

b) Calling superclass constructor

**CODE:**

class Animal {  
 String sound;  
  
 Animal(String sound) {  
 this.sound = sound;  
 }  
}  
  
class Dog extends Animal {  
 Dog(String sound) {  
 super(sound);  
 }  
  
 void displaySound() {  
 System.out.println("Dog Sound: " + super.sound);  
 }  
}  
  
public class SuperKeywordExampleB {  
 public static void main(String[] args) {  
 Dog myDog = new Dog("Woof!");  
 myDog.displaySound();  
 }  
}

OUTPUT:  
Dog Sound :Woof!

c) Invoking superclass method

**CODE:**

class Animal {  
 void eat() {  
 System.out.println("Animal is eating");  
 }  
}  
  
class Dog extends Animal {  
 void eat() {  
 super.eat();  
 System.out.println("Dog is eating");  
 }  
}  
  
public class SuperKeywordExampleC {  
 public static void main(String[] args) {  
 Dog myDog = new Dog();  
 myDog.eat();  
 }  
}

OUTPUT:

Animal is eating

Dog is eating

Generics in Java

13. Single type parameter generic class

**CODE:**

public class Box<T> {  
 private T value;  
  
 // Constructor  
 public Box(T value) {  
 this.value = value;  
 }  
  
 // Getter and Setter  
 public T getValue() {  
 return value;  
 }  
 public void setValue(T value) {  
 this.value = value;  
 }  
  
 public static void main(String[] args) {  
 // Creating a Box of Integer  
 Box<Integer> integerBox = new Box<>(42);  
  
 // Creating a Box of String  
 Box<String> stringBox = new Box<>("Hello, Generics!");  
  
 // Getting values  
 System.out.println("Integer Box Value: " + integerBox.getValue());  
 System.out.println("String Box Value: " + stringBox.getValue());  
  
 // Setting new values  
 integerBox.setValue(99);  
 stringBox.setValue("Updated Value");  
  
 // Getting updated values  
 System.out.println("Updated Integer Box Value: " + integerBox.getValue());  
 System.out.println("Updated String Box Value: " + stringBox.getValue());  
 }  
}

**OUTPUT:**

Integer Box Value: 42

String Box Value: Hello, Generics!

Updated Integer Box Value: 99

Updated String Box Value: Updated Value

14. Multiple type parameter generic class

**CODE:**

public class Pair<K, V> {  
 private K key;  
 private V value;  
  
 // Constructor  
 public Pair(K key, V value) {  
 this.key = key;  
 this.value = value;  
 }  
  
 // Getter and Setter for key  
 public K getKey() {  
 return key;  
 }  
  
 public void setKey(K key) {  
 this.key = key;  
 }  
  
 // Getter and Setter for value  
 public V getValue() {  
 return value;  
 }  
  
 public void setValue(V value) {  
 this.value = value;  
 }  
  
 public static void main(String[] args) {  
 // Creating a Pair with Integer key and String value  
 Pair<Integer, String> pair1 = new Pair<>(1, "One");  
  
 // Creating a Pair with String key and Double value  
 Pair<String, Double> pair2 = new Pair<>("PI", 3.14);  
  
 // Getting key-value pairs  
 System.out.println("Pair 1: Key=" + pair1.getKey() + ", Value=" + pair1.getValue());  
 System.out.println("Pair 2: Key=" + pair2.getKey() + ", Value=" + pair2.getValue());  
  
 // Setting new values  
 pair1.setKey(42);  
 pair2.setValue(2.71);  
  
 // Getting updated key-value pairs  
 System.out.println("Updated Pair 1: Key=" + pair1.getKey() + ", Value=" + pair1.getValue());  
 System.out.println("Updated Pair 2: Key=" + pair2.getKey() + ", Value=" + pair2.getValue());  
 }  
}

**OUTPUT:**

Pair 1: Key=1, Value=One

Pair 2: Key=PI, Value=3.14

Updated Pair 1: Key=42, Value=One

Updated Pair 2: Key=PI, Value=2.71

15. Using generics on methods example

**CODE:**

public class GenericMethodExample {  
  
 // Generic method to compare two values of any type  
 public static <T extends Comparable<T>> boolean isEqual(T value1, T value2) {  
 return value1.compareTo(value2) == 0;  
 }  
  
 // Generic method to find the maximum of an array of any type  
 public static <T extends Comparable<T>> T findMax(T[] array) {  
 if (array == null || array.length == 0) {  
 return null;  
 }  
  
 T max = array[0];  
 for (int i = 1; i < array.length; i++) {  
 if (array[i].compareTo(max) > 0) {  
 max = array[i];  
 }  
 }  
 return max;  
 }  
  
 public static void main(String[] args) {  
 // Comparing two integers  
 int int1 = 5;  
 int int2 = 5;  
 System.out.println("Are integers equal? " + isEqual(int1, int2));  
  
 // Comparing two strings  
 String str1 = "Hello";  
 String str2 = "World";  
 System.out.println("Are strings equal? " + isEqual(str1, str2));  
  
 // Finding the maximum of an array of doubles  
 Double[] doubleArray = { 3.14, 2.71, 1.618 };  
 System.out.println("Maximum double value: " +findMax(doubleArray));  
 }  
}

**OUTPUT:**

Are integers equal? true

Are strings equal? false

Maximum double value: 3.14

16. Restrict use of primitive types using generics

**CODE:**

public class GenericRestrictionExample<T extends Number> {  
  
 private T value;  
  
 public GenericRestrictionExample(T value) {  
 this.value = value;  
 }  
  
 public T getValue() {  
 return value;  
 }  
  
 public static void main(String[] args) {  
 // Using the generic class with Integer  
 GenericRestrictionExample<Integer> integerExample = new GenericRestrictionExample<>(42);  
 System.out.println("Integer value: " + integerExample.getValue());  
  
 // Using the generic class with Double  
 GenericRestrictionExample<Double> doubleExample = new GenericRestrictionExample<>(3.14);  
 System.out.println("Double value: " + doubleExample.getValue());  
  
  
 }  
}

OUTPUT:

Integer Value :42

Double Value :3.14

Java utilities

17. Use Scanner to get Char, String, Int, Float and Double input same moment

CODE:

import java.util.Scanner;  
  
public class InputExample {  
  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
  
 System.out.print("Enter a character: ");  
 char charInput = scanner.next().charAt(0);  
  
 System.out.print("Enter a string: ");  
 String stringInput = scanner.next();  
  
 System.out.print("Enter an integer: ");  
 int intInput = scanner.nextInt();  
  
 System.out.print("Enter a float: ");  
 float floatInput = scanner.nextFloat();  
  
 System.out.print("Enter a double: ");  
 double doubleInput = scanner.nextDouble();

// Displaying the input values  
 System.out.println("Character: " + charInput);  
 System.out.println("String: " + stringInput);  
 System.out.println("Integer: " + intInput);  
 System.out.println("Float: " + floatInput);  
 System.out.println("Double: " + doubleInput);  
  
 // Closing the scanner  
 scanner.close();  
 }  
}

**OUTPUT:**

Enter a character: SUDHARSHAN

Enter a string:

SUDH

Enter an integer: 12

Enter a float: 1.06

Enter a double: 1.77

Character: S

String: SUDH

Integer: 12

Float: 1.06

Double: 1.77

18. Find System Date and Time using Date class

CODE:

import java.util.Date;  
import java.time.LocalDateTime;  
  
public class SystemDateTimeExample {  
  
 public static void main(String[] args) {  
 // Using java.util.Date  
 Date currentDate = new Date();  
 System.out.println(" Current System Date and Time: " + currentDate);  
  
 // Using java.time.LocalDateTime  
 LocalDateTime currentDateTime = LocalDateTime.now();  
 System.out.println("Current System Date and Time: " + currentDateTime);  
 }  
}

**OUTPUT:**

Current System Date and Time: Thu Feb 22 04:43:33 GMT 2024

Current System Date and Time: 2024-02-22T04:43:33.502817

19. Use UUID to generate a random Universally Unique Identifier

**CODE:**

import java.util.UUID;  
  
public class UUIDExample {  
  
 public static void main(String[] args) {  
 // Generating a random UUID  
 UUID uuid = UUID.randomUUID();  
  
 // Displaying the generated UUID  
 System.out.println("Random UUID: " + uuid);  
 }  
}

**OUTPUT:**

Random UUID: a2d1d409-c742-48d3-8537-70ebf02b1ba3

20. Java toString() and equals() method.

CODE:

public class Person {  
 private String name;  
 private int age;  
  
 // Constructor  
 public Person(String name, int age) {  
 this.name = name;  
 this.age = age;  
 }  
  
 // Override toString() method  
 @Override  
 public String toString() {  
 return "Person [name=" + name + ", age=" + age + "]";  
 }  
  
 // Override equals() method  
 @Override  
 public boolean equals(Object obj) {  
 if (this == obj) {  
 return true;  
 }  
 if (obj == null || getClass() != obj.getClass()) {  
 return false;  
 }  
 Person person = (Person) obj;  
 return age == person.age && name.equals(person.name);  
 }  
  
 public static void main(String[] args) {  
 // Creating two Person objects  
 Person person1 = new Person("John", 25);  
 Person person2 = new Person("John", 25);  
  
 // Using toString() method  
 System.out.println("person1: " + person1.toString());  
 System.out.println("person2: " + person2.toString());  
  
 // Using equals() method  
 System.out.println("Are persons equal? " + person1.equals(person2));  
 }  
}

OUTPUT:

person1: Person [name=John, age=25]

person2: Person [name=John, age=25]

Are persons equal? true