Assignment 20.

Part A:

1. Constructor Chaining

Constructor chaining is the process of calling one constructor from another constructor within the same class or from the parent class. It helps in reusing code and avoiding duplication.

There are two ways to perform constructor chaining:

- Using this() – calls another constructor in the same class.
- Using super() – calls the parent class constructor.

Example:

```
class Student {
  Student() {
    System.out.println("Default Constructor");
  }

  Student(String name) {
    this();
    System.out.println("Name: " + name);
  }
}
```

2. Can a Constructor be Overridden?

No, constructors cannot be overridden because constructors are not inherited. Overriding requires inheritance, and since constructors belong only to their class, they cannot be overridden.

However, constructors can be overloaded by defining multiple constructors with different parameter lists within the same class.

3. Difference Between Static and Instance Variables

Static Variable:

- Belongs to the class.
- Only one copy is created for the entire class.
- Declared using the static keyword.

• Accessed using class name.

Instance Variable:

• Belongs to the object.

• Separate copy is created for each object.

• Declared without static keyword.

• Accessed using object reference.

4. Method Hiding

Method hiding occurs when a child class defines a static method with the same name and parameters as a static method in the parent class. Static methods are resolved at compile time, so the method call depends on the reference type.

Example:

```
class Parent {
   static void show() {
      System.out.println("Parent method");
   }
}

class Child extends Parent {
   static void show() {
      System.out.println("Child method");
   }
}
```

5. Limitations of Java

• No multiple inheritance with classes.

• Slower than C/C++ because it runs on JVM.

• Higher memory usage due to JVM and garbage collection.

• No direct pointer manipulation.

• Limited operator overloading (only + for String).

Part B:

1. Program to Demonstrate Nested Try Blocks

```
class NestedTryDemo {
   public static void main(String[] args) {
      try {
         System.out.println("Outer try block");
         int arr[] = {10, 20, 30};

         try {
            System.out.println("Inner try block 1");
            System.out.println(arr[5]);
         } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Handled ArrayIndexOutOfBoundsException");
         }

         try {
            System.out.println("Inner try block 2");
            int x = 10 / 0;
         } catch (ArithmeticException e) {
            System.out.println("Handled ArithmeticException");
         }

      } catch (Exception e) {
         System.out.println("Outer catch block");
      }

      System.out.println("Program continues...");
   }
}
```

2. Program to Demonstrate throw and throws

```
class ThrowThrowsDemo {

   static void checkAge(int age) throws Exception {
      if (age < 18) {
         throw new Exception("Not eligible to vote");
      } else {
         System.out.println("Eligible to vote");
      }
   }

   public static void main(String[] args) {
        try {
            checkAge(16);
        } catch (Exception e) {
            System.out.println("Exception caught: " + e.getMessage());
        }
    }
}
```

3. Program to Show Difference Between Shallow and Deep Copy

```
class Address {
    String city;
    Address(String city) {
        this.city = city;
    }
}

class Person implements Cloneable {
    String name;
    Address address;

    Person(String name, Address address) {
        this.name = name;
        this.address = address;
    }

    protected Object clone() throws CloneNotSupportedException {
        return super.clone();  // Shallow copy
    }

    Person deepCopy() {
        Address newAddress = new Address(address.city);
        return new Person(name, newAddress);
    }
}

class CopyDemo {
    public static void main(String[] args) throws Exception {
        Address addr = new Address("Delhi");
        Person p1 = new Person("Nitin", addr);

        Person p2 = (Person) p1.clone();   // Shallow copy
        Person p3 = p1.deepCopy();         // Deep copy
        p2.address.city = "Mumbai";

        System.out.println("Original City after shallow copy: " + p1.address.city);
        System.out.println("Deep Copy City: " + p3.address.city);
    }
}
```

4. Program to Serialize and Deserialize an Object

```
import java.io.*;

class Student implements Serializable {
    private static final long serialVersionUID = 1L;
    String name;
    int age;

    Student(String name, int age) {
        this.name = name;
        this.age = age;
    }
}

class SerializationDemo {
    public static void main(String[] args) throws Exception {

        Student s1 = new Student("Nitin", 22);

        FileOutputStream fos = new FileOutputStream("student.ser");
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject(s1);
        oos.close();
        fos.close();

        FileInputStream fis = new FileInputStream("student.ser");
        ObjectInputStream ois = new ObjectInputStream(fis);
        Student s2 = (Student) ois.readObject();
        ois.close();
        fis.close();

        System.out.println("Name: " + s2.name);
        System.out.println("Age: " + s2.age);
    }
}
```

5. Program to Use Comparable to Sort Custom Objects

```
import java.util.*;

class Employee implements Comparable<Employee> {
    int id;
    String name;

    Employee(int id, String name) {
        this.id = id;
        this.name = name;
    }

    public int compareTo(Employee e) {
        return this.id - e.id;
    }

    public String toString() {
        return id + " " + name;
    }
}

class ComparableDemo {
    public static void main(String[] args) {
        List<Employee> list = new ArrayList<>();

        list.add(new Employee(3, "Amit"));
        list.add(new Employee(1, "Nitin"));
        list.add(new Employee(2, "Rahul"));

        Collections.sort(list);

        for (Employee e : list) {
            System.out.println(e);
        }
    }
}
```