1)Inheritance and Polymorpism  
  
class Vehicle {

    String make, model;

    Vehicle(String make, String model) {

        this.make = make;

        this.model = model;

    }

    void startEngine() {

        System.out.println("Starting the vehicle engine...");

    }

}

class Car extends Vehicle {

    Car(String make, String model) {

        super(make, model);

    }

    @Override

    void startEngine() {

        System.out.println(make + " " + model + " car engine started with a key ignition.");

    }

}

class Motorcycle extends Vehicle {

    Motorcycle(String make, String model) {

        super(make, model);

    }

    @Override

    void startEngine() {

        System.out.println(make + " " + model + " motorcycle engine started with a self-start button.");

    }

}

public class Main {

    public static void main(String[] args) {

        Vehicle v1 = new Car("Toyota", "Corolla");

        Vehicle v2 = new Motorcycle("Yamaha", "R15");

        v1.startEngine();

        v2.startEngine();

    }

}

2)Abstraction and Interface  
  
class Vehicle {

    String make, model;

    Vehicle(String make, String model) {

        this.make = make;

        this.model = model;

    }

    void startEngine() {

        System.out.println("Starting the vehicle engine...");

    }

}

class Car extends Vehicle {

    Car(String make, String model) {

        super(make, model);

    }

    @Override

    void startEngine() {

        System.out.println(make + " " + model + " car engine started with a key ignition.");

    }

}

class Motorcycle extends Vehicle {

    Motorcycle(String make, String model) {

        super(make, model);

    }

    @Override

    void startEngine() {interface Shape {

        double calculateArea();

        void displayShapeInfo();

    }

    class Circle implements Shape {

        private double radius;

        Circle(double radius) {

            this.radius = radius;

        }

        @Override

        public double calculateArea() {

            return Math.PI \* radius \* radius;

        }

        @Override

        public void displayShapeInfo() {

            System.out.println("Shape: Circle | Radius: " + radius + " | Area: " + calculateArea());

        }

    }

    class Rectangle implements Shape {

        private double length, width;

        Rectangle(double length, double width) {

            this.length = length;

            this.width = width;

        }

        @Override

        public double calculateArea() {

            return length \* width;

        }

        @Override

        public void displayShapeInfo() {

            System.out.println("Shape: Rectangle | Length: " + length + ", Width: " + width + " | Area: " + calculateArea());

        }

    }

    public class Main {

        public static void main(String[] args) {

            Shape s1 = new Circle(5);

            Shape s2 = new Rectangle(4, 6);

            s1.displayShapeInfo();

            s2.displayShapeInfo();

        }

    }

        System.out.println(make + " " + model + " motorcycle engine started with a self-start button.");

    }

}

public class Main {

    public static void main(String[] args) {

        Vehicle v1 = new Car("Toyota", "Corolla");

        Vehicle v2 = new Motorcycle("Yamaha", "R15");

        v1.startEngine();

        v2.startEngine();

    }

}

3)Abstraction classes and Polymorphism  
  
abstract class Animal {

protected String name;

Animal(String name) {

this.name = name;

}

abstract void makeSound();

}

class Dog extends Animal {

Dog(String name) {

super(name);

}

@Override

void makeSound() {

System.out.println(name + " says: Woof Woof!");

}

}

class Cat extends Animal {

Cat(String name) {

super(name);

}

@Override

void makeSound() {

System.out.println(name + " says: Meow Meow!");

}

}

public class Main {

public static void main(String[] args) {

Animal[] animals = {

new Dog("Buddy"),

new Cat("Whiskers"),

new Dog("Max"),

new Cat("Luna")

};

for (Animal a : animals) {

a.makeSound();

}

}

}  
  
4)Interface and Encapsulation  
  
interface Logger {

void logInfo(String message);

void logError(String message);

}

class FileLogger implements Logger {

@Override

public void logInfo(String message) {

System.out.println("[File] INFO: " + message);

}

@Override

public void logError(String message) {

System.out.println("[File] ERROR: " + message);

}

}

class DatabaseLogger implements Logger {

@Override

public void logInfo(String message) {

System.out.println("[Database] INFO: " + message);

}

@Override

public void logError(String message) {

System.out.println("[Database] ERROR: " + message);

}

}

class Application {

private Logger logger;

Application(Logger logger) {

this.logger = logger;

}

void performApplicationTask() {

logger.logInfo("Task started");

try {

int result = 10 / 2; // Simulated task

logger.logInfo("Task completed successfully. Result: " + result);

} catch (Exception e) {

logger.logError("Task failed: " + e.getMessage());

}

}

}

public class Main {

public static void main(String[] args) {

Application appFileLogger = new Application(new FileLogger());

appFileLogger.performApplicationTask();

System.out.println();

Application appDbLogger = new Application(new DatabaseLogger());

appDbLogger.performApplicationTask();

}

}  
  
  
5)Exception Handling  
  
class InvalidDataException extends Exception {

    public InvalidDataException(String message) {

        super(message);

    }

}

class DataProcessor {

    public double processData(int[] data) throws InvalidDataException {

        if (data == null || data.length == 0) {

            throw new InvalidDataException("Data array is empty or null");

        }

        int sum = 0;

        for (int num : data) {

            sum += num;

        }

        return (double) sum / data.length;

    }

}

public class Main {

    public static void main(String[] args) {

        DataProcessor processor = new DataProcessor();

        int[] validData = {10, 20, 30, 40, 50};

        int[] emptyData = {};

        try {

            double avg = processor.processData(validData);

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

        } catch (InvalidDataException e) {

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

        }

        try {

            double avg = processor.processData(emptyData);

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

        } catch (InvalidDataException e) {

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

        }

    }

}