

1. Implement Abstract class with overloading and overriding

CODE:

```
package com.task;

abstract class Animals {
    // Abstract method
    abstract void sound();

    // Concrete method
    void sleep() {
        System.out.println("The animal is sleeping.");
    }

    // Overloaded method
    void eat() {
        System.out.println("The animal is eating.");
    }

    // Overloaded method with different parameters
    void eat(String food) {
        System.out.println("The animal is eating " + food);
    }
}

class Dog extends Animals {
    // Overriding the abstract method
    @Override
    void sound() {
        System.out.println("The dog barks.");
    }

    // Overriding the concrete method
    @Override
    void sleep() {
        System.out.println("The dog is sleeping.");
    }
}

public class Animal {
```

```

public static void main(String[] args) {
    Animals myDog = new Dog();
    myDog.sound(); // Outputs: The dog barks.

    myDog.sleep(); // Outputs: The dog is sleeping.
    myDog.eat(); // Outputs: The animal is eating.
    myDog.eat("bone"); // Outputs: The animal is eating bone.
}
}

```

OUTPUT:

The screenshot shows the Eclipse IDE interface. The Package Explorer on the left displays the project structure, including the 'com.task' package containing 'Animal.java'. The main editor window shows the source code of 'Animal.java', which defines an abstract class 'Animals' with methods 'sound()', 'sleep()', and 'eat()'. The 'main' method in the 'com.task' package instantiates a 'Dog' object (which inherits from 'Animals') and calls these methods. The Console window at the bottom shows the output of the program: 'The dog barks.', 'The dog is sleeping.', 'The animal is eating.', and 'The animal is eating bone'.

```

1 package com.task;
2
3 abstract class Animals {
4     // Abstract method
5     abstract void sound();
6
7     // Concrete method
8     void sleep() {
9         System.out.println("The animal is sleeping.");
10    }
11
12    // Overloaded method
13    void eat() {
14        System.out.println("The animal is eating.");
15    }
16
17    // Overloaded method with different parameters
18    void eat(String food) {
19        System.out.println("The animal is eating " + food);
20    }
21 }

```

```

<terminated> Animal [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (9 Aug 2024, 7:41:40 pm - 7:41:41 pm) [pid: 3896]
The dog barks.
The dog is sleeping.
The animal is eating.
The animal is eating bone

```

2. Implement Multiple inheritance with Interface

CODE:

```
package com.task;

interface CanFly {
    void fly();
}

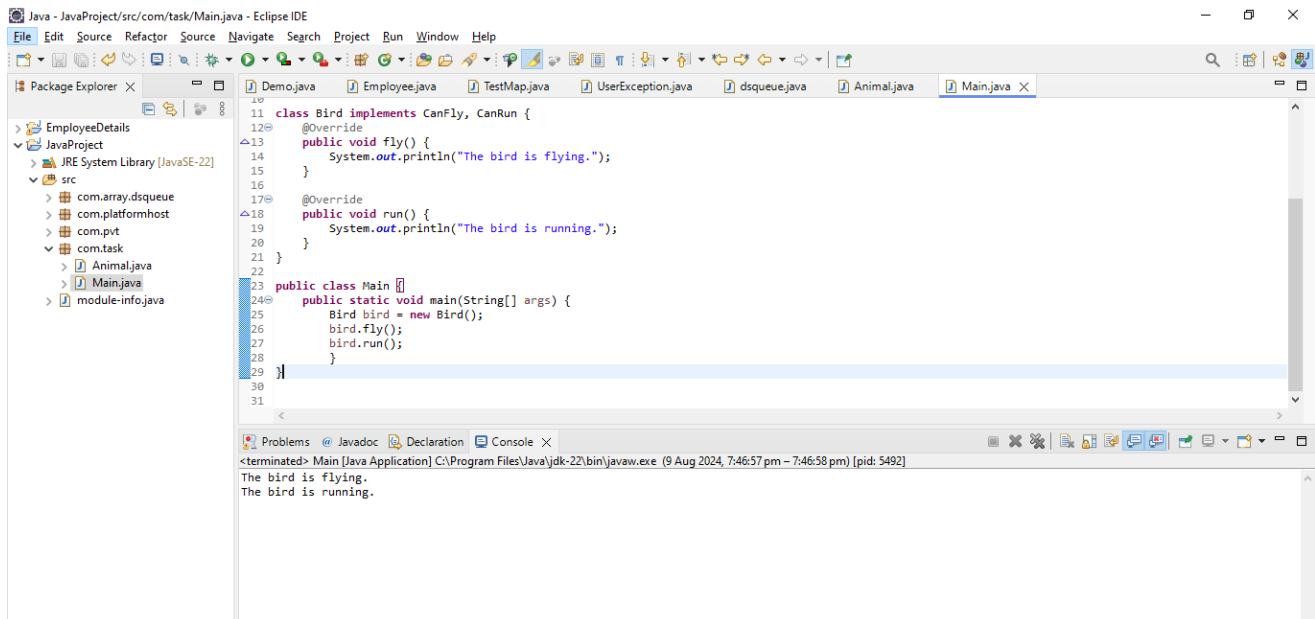
interface CanRun {
    void run();
}

class Bird implements CanFly, CanRun {
    @Override
    public void fly() {
        System.out.println("The bird is flying.");
    }

    @Override
    public void run() {
        System.out.println("The bird is running.");
    }
}

public class Main {
    public static void main(String[] args) {
        Bird bird = new Bird();
        bird.fly();
        bird.run();
    }
}
```

OUTPUT:



3. Show final methods in the class that can't be overridden

CODE:

```
package com.task;
```

```
class Vehicle {  
    // Final method  
    final void start() {  
        System.out.println("The vehicle is starting.");  
    }  
  
    void stop() {  
        System.out.println("The vehicle is stopping.");  
    }  
}
```

```
class Car extends Vehicle {  
    // Trying to override the final method would cause a compile-time error  
    // @Override  
    // void start() {  
    //     System.out.println("The car is starting.");  
    // }
```

```

@Override
void stop() {
    System.out.println("The car is stopping.");
}
}

```

```

public class Main {
    public static void main(String[] args) {
        Car myCar = new Car();
        myCar.start();
        myCar.stop();
    }
}

```

OUTPUT:

The screenshot shows the Eclipse IDE interface. The Package Explorer on the left displays the project structure, including the 'com.task' package with files 'Animal.java' and 'Main.java'. The main editor window shows the source code for 'Main.java', which defines a 'Vehicle' class with 'start()' and 'stop()' methods, and a 'Car' class that extends 'Vehicle' and overrides these methods. The Console window at the bottom shows the output of the program: 'The vehicle is starting.' followed by 'The car is stopping.'.

```

1 package com.task;
2
3 class Vehicle {
4     // Final method
5     final void start() {
6         System.out.println("The vehicle is starting.");
7     }
8
9     void stop() {
10        System.out.println("The vehicle is stopping.");
11    }
12 }
13
14 class Car extends Vehicle {
15     // Trying to override the final method would cause a compile-time error
16     // @Override
17     void start() {
18         // System.out.println("The car is starting.");
19     }
20
21     @Override
22     void stop() {
23         System.out.println("The car is stopping.");
24     }
25 }

```

Console Output:

```

<terminated> Main [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (9 Aug 2024, 7:51:56 pm - 7:51:57 pm) [pid: 5856]
The vehicle is starting.
The car is stopping.

```