Problem 1: Using Abstraction and Interface Together

Problem: Create an abstract class `Appliance` with an abstract method `turnOn()`. Create an interface `Adjustable` with a method `adjust()`. Implement two classes `Fan` and `AirConditioner` that extend `Appliance` and implement `Adjustable`. Write a program to demonstrate the behavior of both classes.

Solution:

```
java
Copy code
// Abstract class Appliance
abstract class Appliance {
    abstract void turnOn();  // Abstract method
}

// Interface Adjustable
interface Adjustable {
    void adjust();  // Interface method
}

// Fan class extending Appliance and implementing Adjustable
class Fan extends Appliance implements Adjustable {
    @Override
    public void turnOn() {
        System.out.println("Fan is turned on.");
    }

    @Override
    public void adjust() {
        System.out.println("Fan speed is adjusted.");
    }
}

// AirConditioner class extending Appliance and implementing Adjustable
class AirConditioner extends Appliance implements Adjustable {
    @Override
    public void turnOn() {
        System.out.println("Air conditioner is turned on.");
    }

    @Override
    public void adjust() {
        System.out.println("Air conditioner temperature is adjusted.");
    }
}

public class Main {
    public static void main(String[] args) {
        Appliance fan = new Fan();
        fan.turnOn();
        ((Fan) fan).adjust();  // Type cast to access adjustable
functionality
```

```
        Appliance ac = new AirConditioner();
        ac.turnOn();
        ((AirConditioner) ac).adjust();
    }
}
```

Output:

```
csharp
Copy code
Fan is turned on.
Fan speed is adjusted.
Air conditioner is turned on.
Air conditioner temperature is adjusted.
```

Problem 2: Multiple Interface Inheritance

Problem: Create two interfaces, `Engine` and `Gear`, with methods `startEngine()` and `changeGear()`, respectively. Then create a class `Car` that implements both interfaces and provide concrete implementations for both methods.

Solution:

```
java
Copy code
// Interface Engine
interface Engine {
    void startEngine();  // Interface method
}

// Interface Gear
interface Gear {
    void changeGear();  // Interface method
}

// Car class implementing both Engine and Gear interfaces
class Car implements Engine, Gear {
    @Override
    public void startEngine() {
        System.out.println("Car engine started.");
    }

    @Override
    public void changeGear() {
        System.out.println("Car gear changed.");
    }
}

public class Main {
    public static void main(String[] args) {
        Car myCar = new Car();
        myCar.startEngine();  // Calling Engine method
        myCar.changeGear();   // Calling Gear method
```

```
    }
}
```

Output:

```
Copy code
Car engine started.
Car gear changed.
```

Problem 3: Abstract Class and Interface Combination

Problem: Create an abstract class `Computer` with an abstract method `bootUp()`. Create an interface `Upgradeable` with a method `upgrade()`. Implement two classes `Laptop` and `Desktop` that extend `Computer` and implement `Upgradeable`. Demonstrate how each class handles booting up and upgrading.

Solution:

```
java
Copy code
// Abstract class Computer
abstract class Computer {
    abstract void bootUp();  // Abstract method
}

// Interface Upgradeable
interface Upgradeable {
    void upgrade();  // Interface method
}

// Laptop class extending Computer and implementing Upgradeable
class Laptop extends Computer implements Upgradeable {
    @Override
    public void bootUp() {
        System.out.println("Laptop is booting up.");
    }

    @Override
    public void upgrade() {
        System.out.println("Laptop upgraded.");
    }
}

// Desktop class extending Computer and implementing Upgradeable
class Desktop extends Computer implements Upgradeable {
    @Override
    public void bootUp() {
        System.out.println("Desktop is booting up.");
    }

    @Override
    public void upgrade() {
```

```
        System.out.println("Desktop upgraded.");
    }
}

public class Main {
    public static void main(String[] args) {
        Computer laptop = new Laptop();
        laptop.bootUp();
        ((Laptop) laptop).upgrade();  // Casting to access the upgrade method

        Computer desktop = new Desktop();
        desktop.bootUp();
        ((Desktop) desktop).upgrade();
    }
}
```

Output:

```
csharp
Copy code
Laptop is booting up.
Laptop upgraded.
Desktop is booting up.
Desktop upgraded.
```

Problem 4: Interface with Multiple Implementations

Problem: Create an interface `Payment` with a method `makePayment()`. Then create two classes `CreditCardPayment` and `PayPalPayment` that implement the `Payment` interface. Each class should provide its specific implementation of `makePayment()`. Write a program to demonstrate how different payment methods are handled.

Solution:

```
java
Copy code
// Interface Payment
interface Payment {
    void makePayment(double amount);  // Interface method
}

// CreditCardPayment class implementing Payment interface
class CreditCardPayment implements Payment {
    @Override
    public void makePayment(double amount) {
        System.out.println("Payment of $" + amount + " made via Credit
Card.");
    }
}

// PayPalPayment class implementing Payment interface
class PayPalPayment implements Payment {
```

```
    @Override
    public void makePayment(double amount) {
        System.out.println("Payment of $" + amount + " made via PayPal.");
    }
}

public class Main {
    public static void main(String[] args) {
        Payment creditCardPayment = new CreditCardPayment();
        creditCardPayment.makePayment(100.0);

        Payment payPalPayment = new PayPalPayment();
        payPalPayment.makePayment(200.0);
    }
}
```

Output:

```
bash
Copy code
Payment of $100.0 made via Credit Card.
Payment of $200.0 made via PayPal.
```

Problem 5: Interface Default Methods

Problem: Create an interface `Printer` with a default method `showBrand()` that prints "Generic Printer Brand". Create a class `HPPrinter` that implements the `Printer` interface and override the `showBrand()` method to print "HP Printer". Also, create a class `CanonPrinter` that doesn't override `showBrand()` and uses the default method.

Solution:

```
java
Copy code
// Interface Printer
interface Printer {
    default void showBrand() {
        System.out.println("Generic Printer Brand");
    }

    void printDocument(String document);  // Interface method
}

// HPPrinter class implementing Printer interface and overriding showBrand()
class HPPrinter implements Printer {
    @Override
    public void showBrand() {
        System.out.println("HP Printer");
    }

    @Override
    public void printDocument(String document) {
```

```
        System.out.println("Printing from HP: " + document);
    }
}

// CanonPrinter class implementing Printer interface but using default
showBrand()
class CanonPrinter implements Printer {
    @Override
    public void printDocument(String document) {
        System.out.println("Printing from Canon: " + document);
    }
}

public class Main {
    public static void main(String[] args) {
        Printer hp = new HPPrinter();
        hp.showBrand();
        hp.printDocument("HP Document");

        Printer canon = new CanonPrinter();
        canon.showBrand();  // Using default method from the interface
        canon.printDocument("Canon Document");
    }
}
```

Output:

```
python
Copy code
HP Printer
Printing from HP: HP Document
Generic Printer Brand
Printing from Canon: Canon Document
```