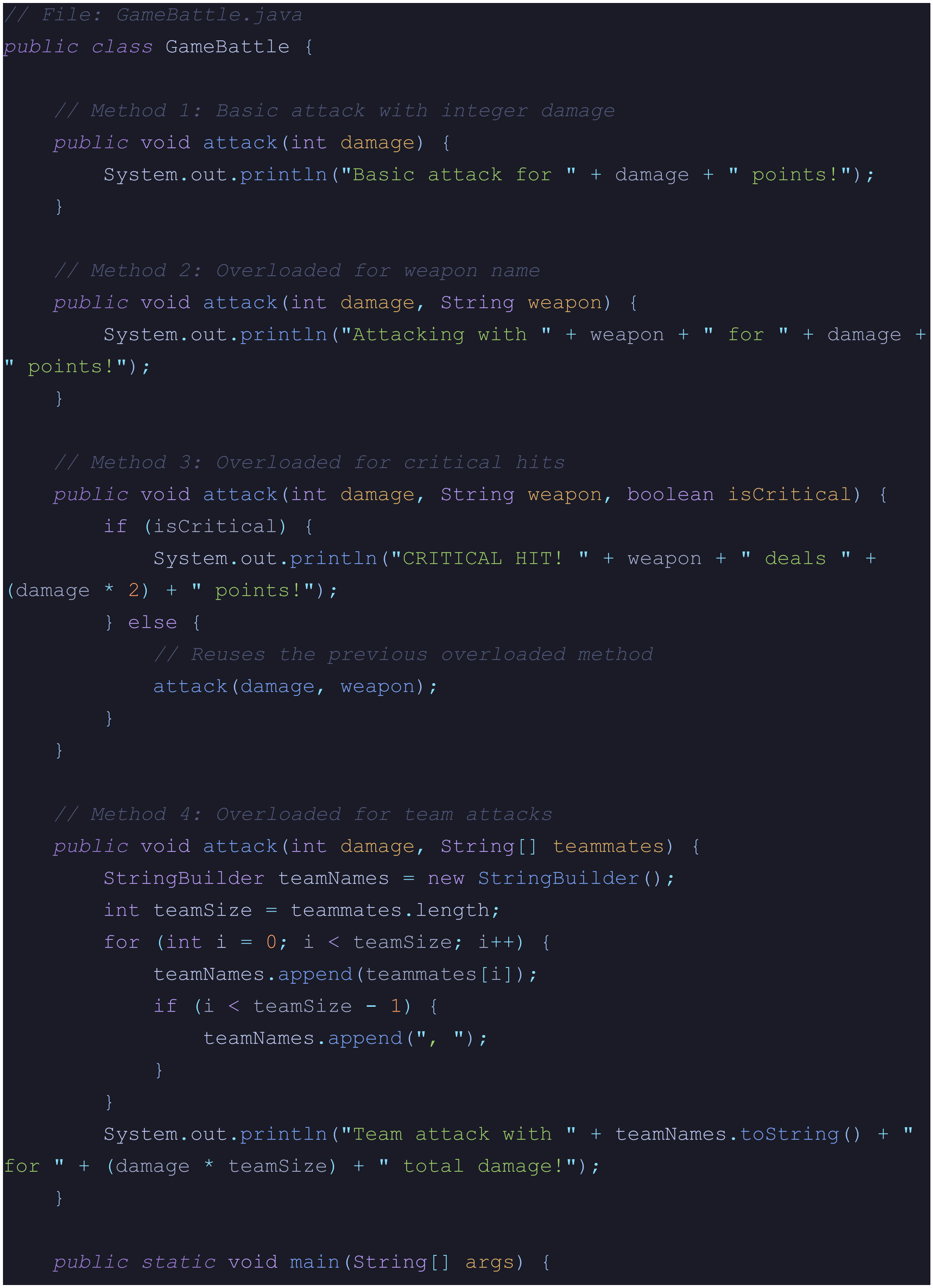
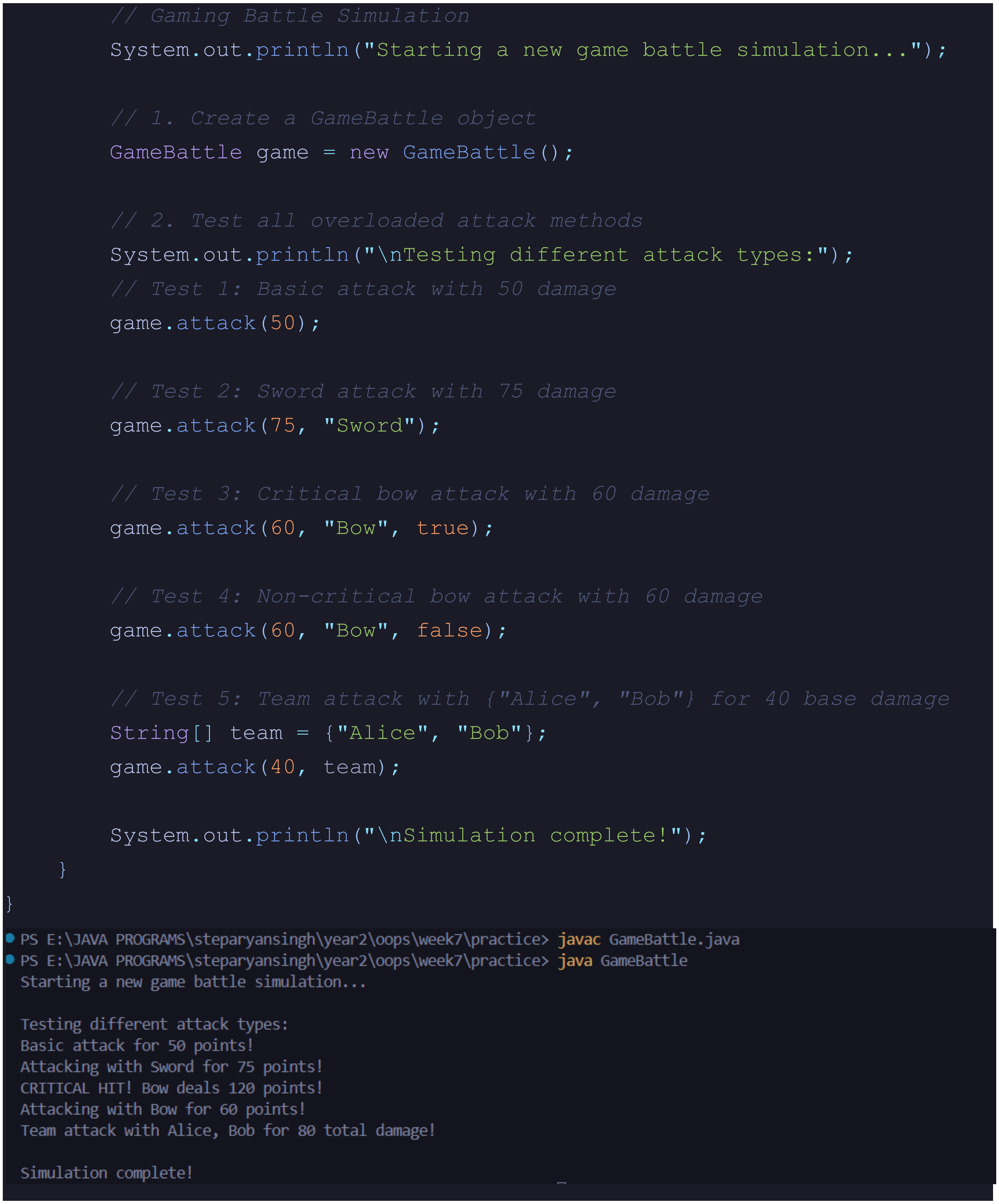
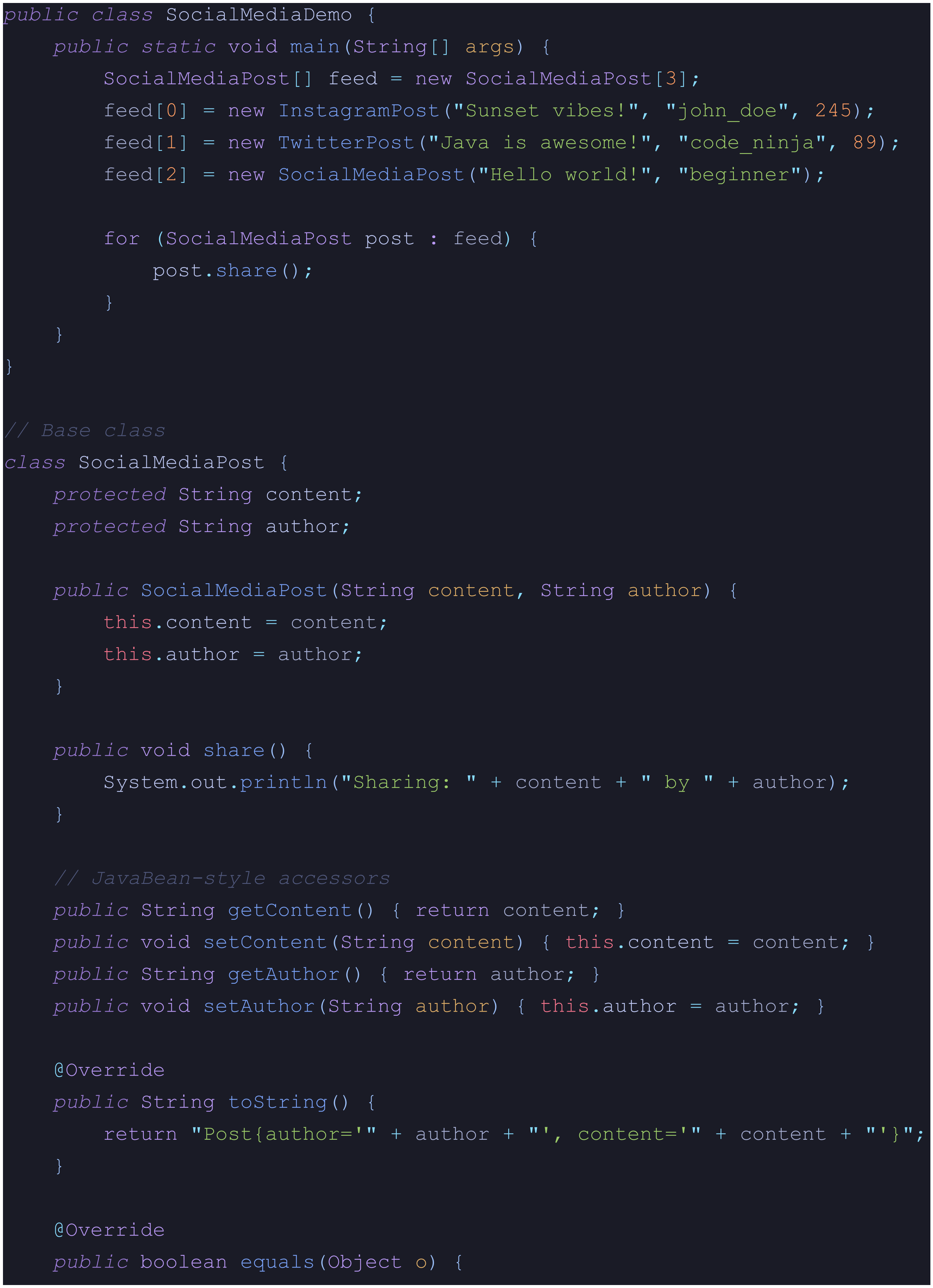


```
// File: GameBattle.java
public class GameBattle {

    // Method 1: Basic attack with integer damage
    public void attack(int damage) {
        System.out.println("Basic attack for " + damage + " points!");
    }

    // Method 2: Overloaded for weapon name
    public void attack(int damage, String weapon) {
        System.out.println("Attacking with " + weapon + " for " + damage +
" points!");
    }

    // Method 3: Overloaded for critical hits
    public void attack(int damage, String weapon, boolean isCritical) {
        if (isCritical) {
            System.out.println("CRITICAL HIT! " + weapon + " deals " +
(damage * 2) + " points!");
        } else {
            // Reuses the previous overloaded method
            attack(damage, weapon);
        }
    }

    // Method 4: Overloaded for team attacks
    public void attack(int damage, String[] teammates) {
        StringBuilder teamNames = new StringBuilder();
        int teamSize = teammates.length;
        for (int i = 0; i < teamSize; i++) {
            teamNames.append(teammates[i]);
            if (i < teamSize - 1) {
                teamNames.append(", ");
            }
        }
        System.out.println("Team attack with " + teamNames.toString() + "
for " + (damage * teamSize) + " total damage!");
    }

    public static void main(String[] args) {
```

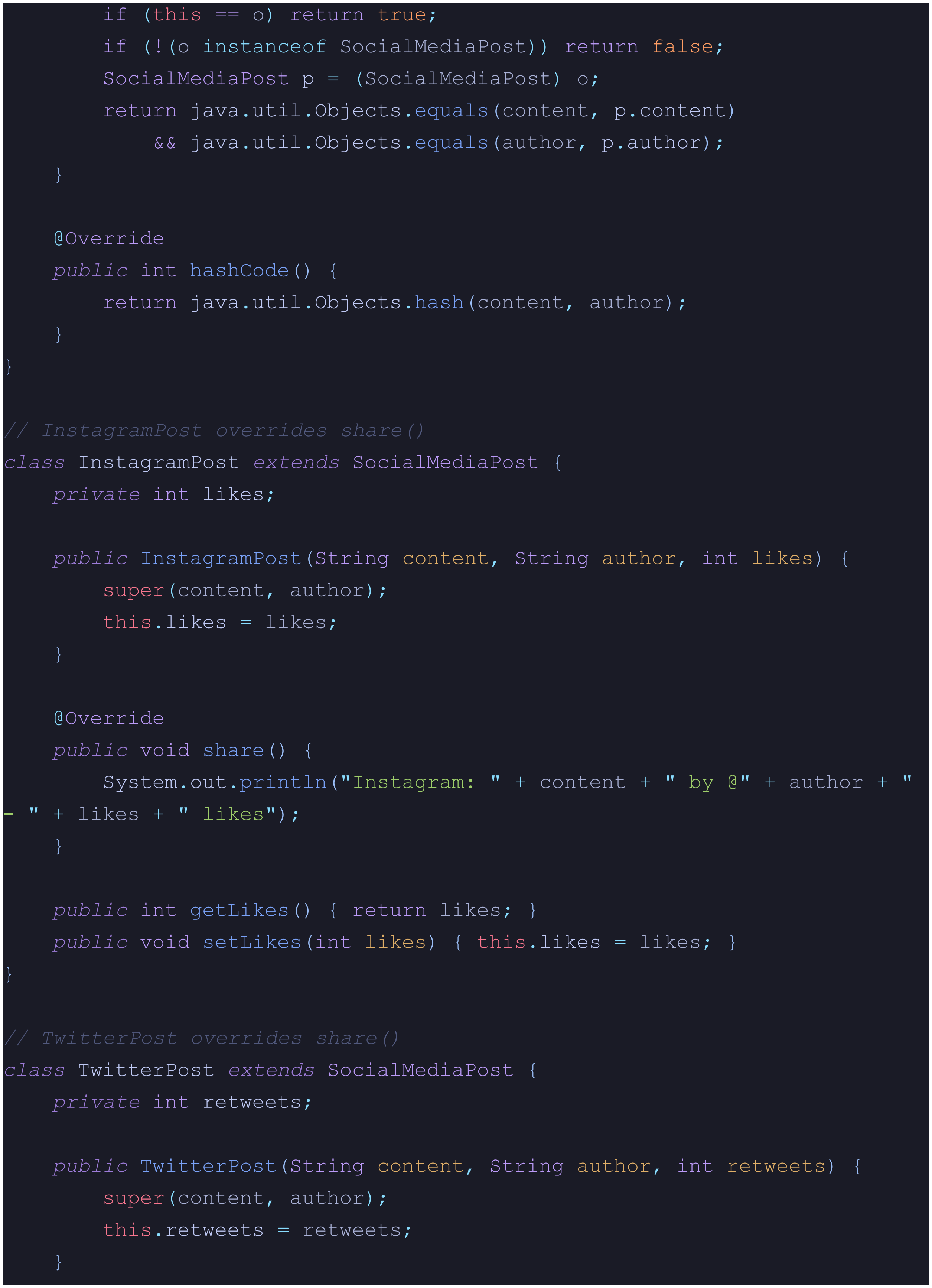


```java
        // Gaming Battle Simulation
        System.out.println("Starting a new game battle simulation...");

        // 1. Create a GameBattle object
        GameBattle game = new GameBattle();

        // 2. Test all overloaded attack methods
        System.out.println("\nTesting different attack types:");
        // Test 1: Basic attack with 50 damage
        game.attack(50);

        // Test 2: Sword attack with 75 damage
        game.attack(75, "Sword");

        // Test 3: Critical bow attack with 60 damage
        game.attack(60, "Bow", true);

        // Test 4: Non-critical bow attack with 60 damage
        game.attack(60, "Bow", false);

        // Test 5: Team attack with {"Alice", "Bob"} for 40 base damage
        String[] team = {"Alice", "Bob"};
        game.attack(40, team);

        System.out.println("\nSimulation complete!");
    }
}
```

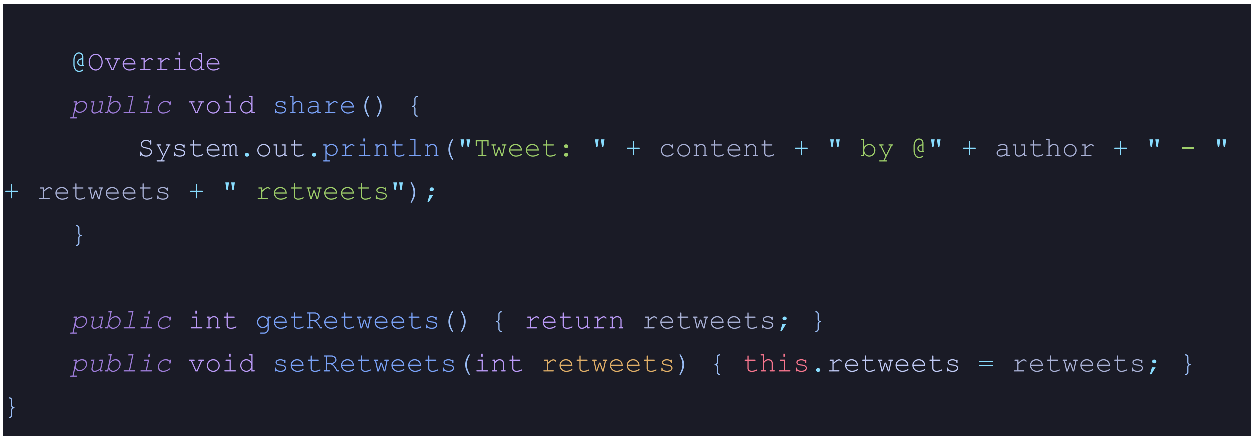
```
PS E:\JAVA PROGRAMS\stepar yansingh\year2\oops\week7\practice> javac GameBattle.java
PS E:\JAVA PROGRAMS\steparyansingh\year2\oops\week7\practice> java GameBattle
Starting a new game battle simulation...

Testing different attack types:
Basic attack for 50 points!
Attacking with Sword for 75 points!
CRITICAL HIT! Bow deals 120 points!
Attacking with Bow for 60 points!
Team attack with Alice, Bob for 80 total damage!

Simulation complete!
```



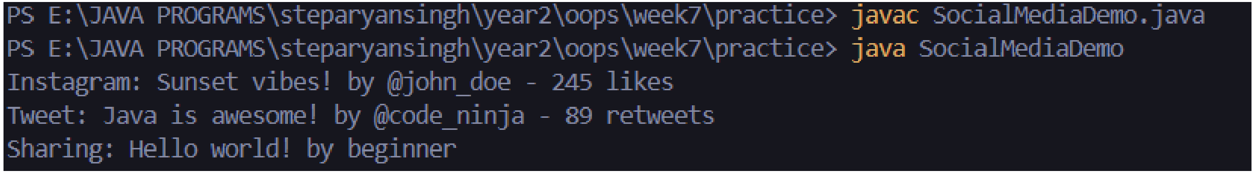
```java
public class SocialMediaDemo {
    public static void main(String[] args) {
        SocialMediaPost[] feed = new SocialMediaPost[3];
        feed[0] = new InstagramPost("Sunset vibes!", "john_doe", 245);
        feed[1] = new TwitterPost("Java is awesome!", "code_ninja", 89);
        feed[2] = new SocialMediaPost("Hello world!", "beginner");

        for (SocialMediaPost post : feed) {
            post.share();
        }
    }
}

// Base class
class SocialMediaPost {
    protected String content;
    protected String author;

    public SocialMediaPost(String content, String author) {
        this.content = content;
        this.author = author;
    }

    public void share() {
        System.out.println("Sharing: " + content + " by " + author);
    }

    // JavaBean-style accessors
    public String getContent() { return content; }
    public void setContent(String content) { this.content = content; }
    public String getAuthor() { return author; }
    public void setAuthor(String author) { this.author = author; }

    @Override
    public String toString() {
        return "Post{author='" + author + "', content='" + content + "'}";
    }

    @Override
    public boolean equals(Object o) {
```



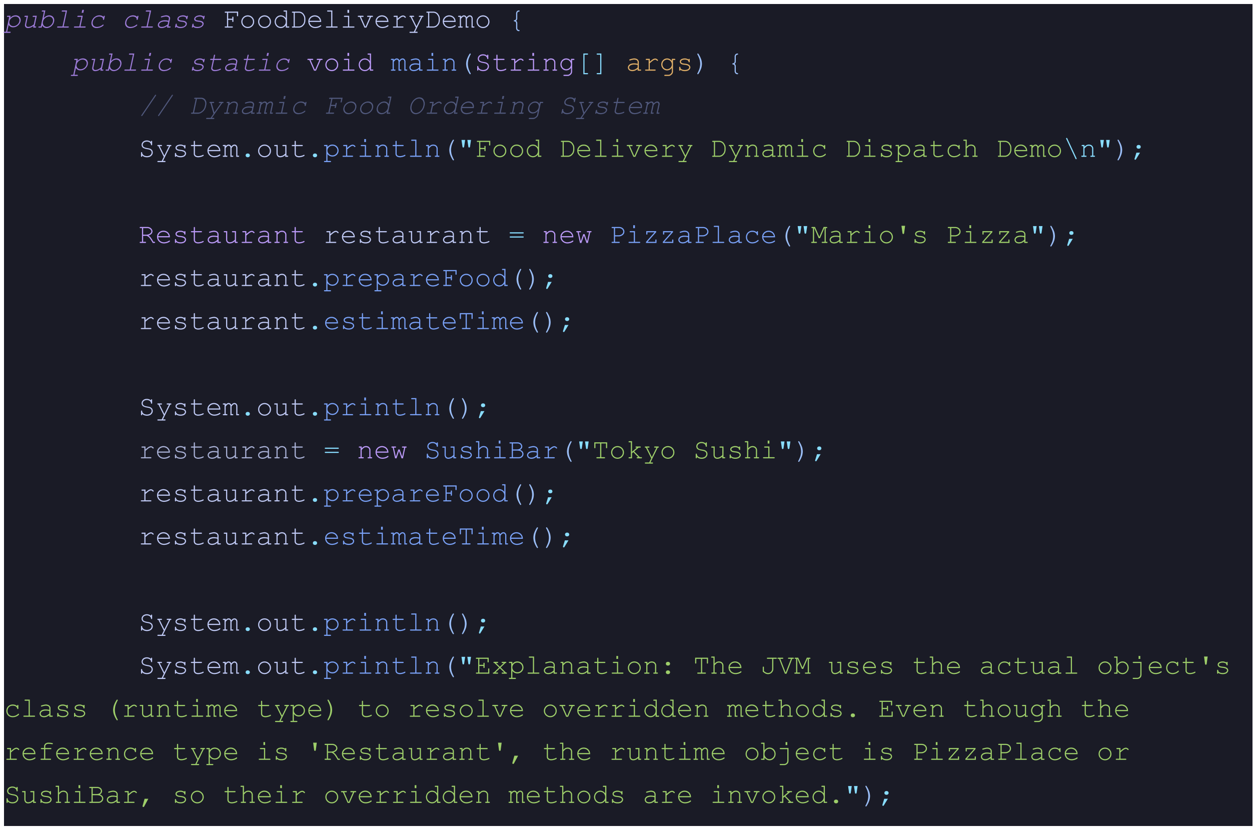
```java
        if (this == o) return true;
        if (!(o instanceof SocialMediaPost)) return false;
        SocialMediaPost p = (SocialMediaPost) o;
        return java.util.Objects.equals(content, p.content)
            && java.util.Objects.equals(author, p.author);
    }

    @Override
    public int hashCode() {
        return java.util.Objects.hash(content, author);
    }
}

// InstagramPost overrides share()
class InstagramPost extends SocialMediaPost {
    private int likes;

    public InstagramPost(String content, String author, int likes) {
        super(content, author);
        this.likes = likes;
    }

    @Override
    public void share() {
        System.out.println("Instagram: " + content + " by @" + author + "
- " + likes + " likes");
    }

    public int getLikes() { return likes; }
    public void setLikes(int likes) { this.likes = likes; }
}

// TwitterPost overrides share()
class TwitterPost extends SocialMediaPost {
    private int retweets;

    public TwitterPost(String content, String author, int retweets) {
        super(content, author);
        this.retweets = retweets;
    }
```



```
    @Override
    public void share() {
        System.out.println("Tweet: " + content + " by @" + author + " - "
+ retweets + " retweets");
    }

    public int getRetweets() { return retweets; }
    public void setRetweets(int retweets) { this.retweets = retweets; }
}
```

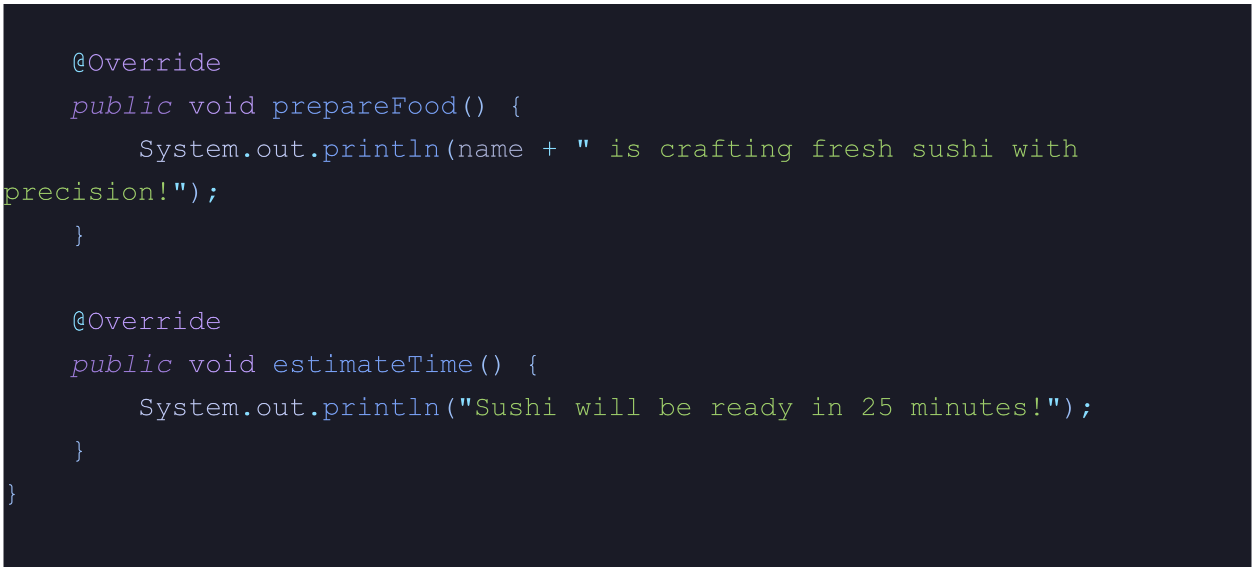


```
PS E:\JAVA PROGRAMS\stepariyansingh\year2\oops\week7\practice> javac SocialMediaDemo.java
PS E:\JAVA PROGRAMS\stepariyansingh\year2\oops\week7\practice> java SocialMediaDemo
Instagram: Sunset vibes! by @john_doe - 245 likes
Tweet: Java is awesome! by @code_ninja - 89 retweets
Sharing: Hello world! by beginner
```

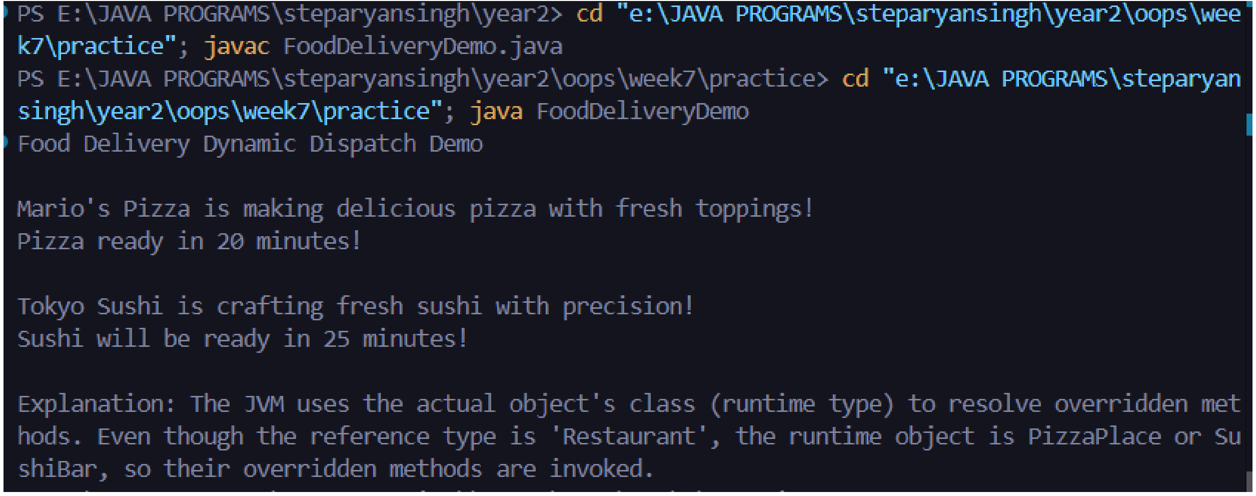


```
public class FoodDeliveryDemo {
    public static void main(String[] args) {
        // Dynamic Food Ordering System
        System.out.println("Food Delivery Dynamic Dispatch Demo\n");

        Restaurant restaurant = new PizzaPlace("Mario's Pizza");
        restaurant.prepareFood();
        restaurant.estimateTime();

        System.out.println();
        restaurant = new SushiBar("Tokyo Sushi");
        restaurant.prepareFood();
        restaurant.estimateTime();

        System.out.println();
        System.out.println("Explanation: The JVM uses the actual object's
class (runtime type) to resolve overridden methods. Even though the
reference type is 'Restaurant', the runtime object is PizzaPlace or
SushiBar, so their overridden methods are invoked.");
```

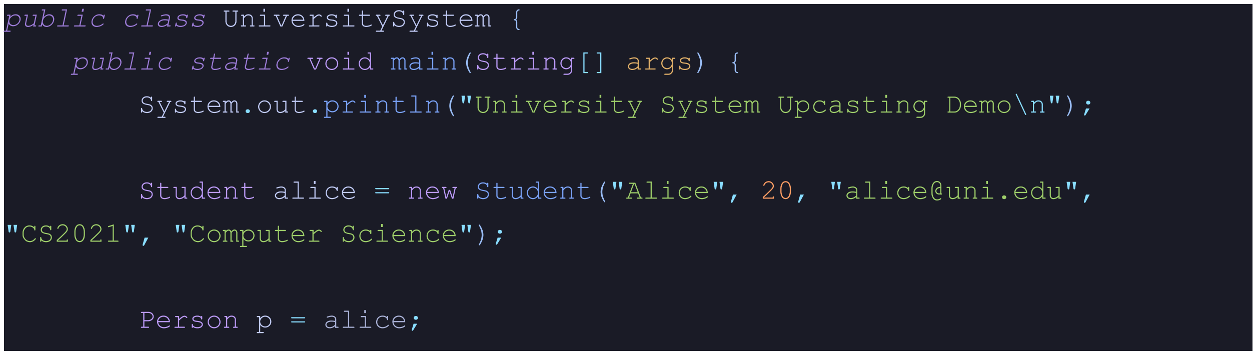
```
    }
}

class Restaurant {
    protected String name;

    public Restaurant(String name) {
        this.name = name;
    }

    public void prepareFood() {
        System.out.println(name + " is preparing generic food");
    }

    public void estimateTime() {
        System.out.println("Estimated time: 30 minutes");
    }
}

class PizzaPlace extends Restaurant {
    public PizzaPlace(String name) {
        super(name);
    }

    @Override
    public void prepareFood() {
        System.out.println(name + " is making delicious pizza with fresh
toppings!");
    }

    @Override
    public void estimateTime() {
        System.out.println("Pizza ready in 20 minutes!");
    }
}

class SushiBar extends Restaurant {
    public SushiBar(String name) {
        super(name);
    }
```



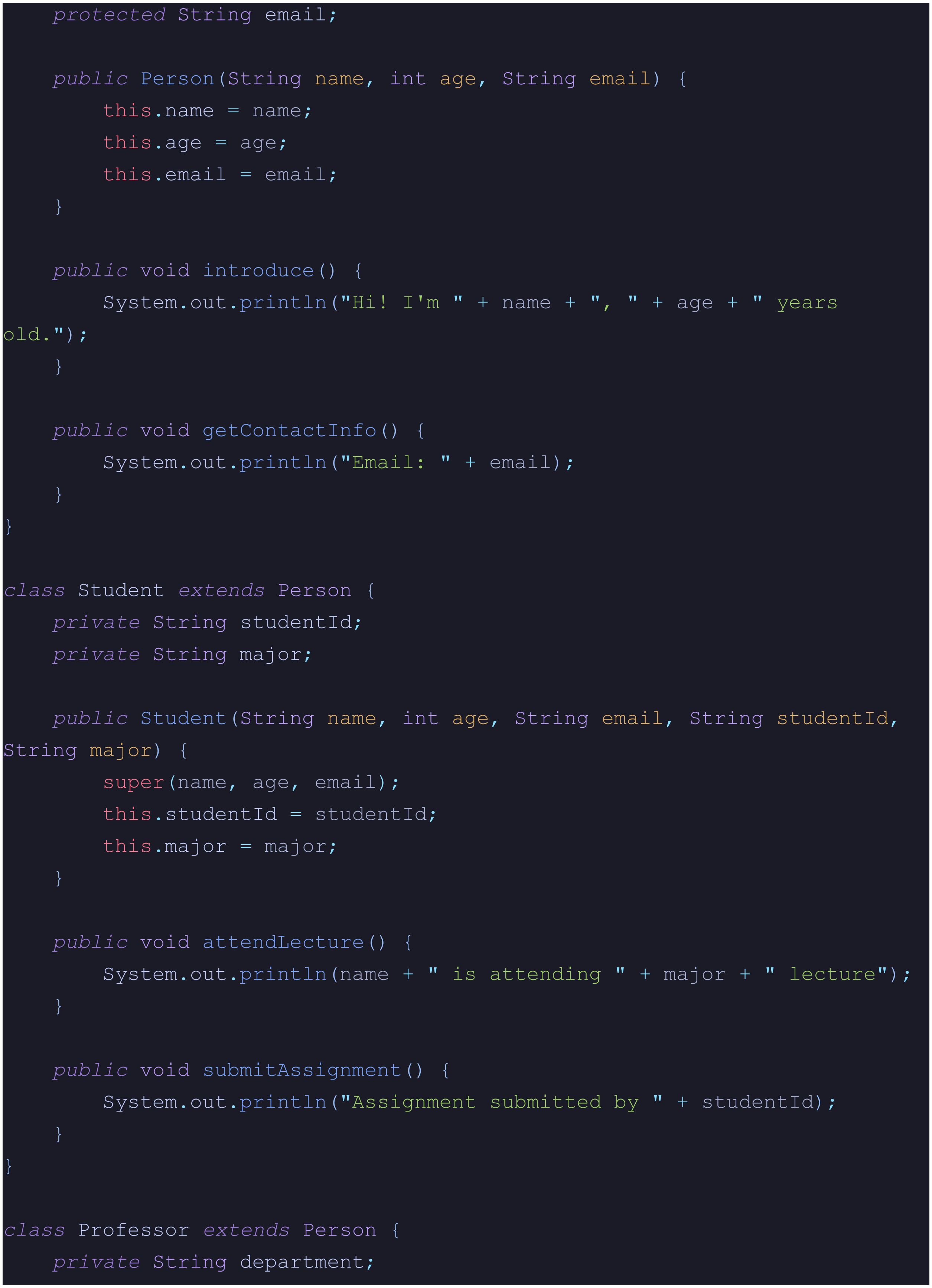
```
    @Override
    public void prepareFood() {
        System.out.println(name + " is crafting fresh sushi with 
precision!");
    }

    @Override
    public void estimateTime() {
        System.out.println("Sushi will be ready in 25 minutes!");
    }
}
```



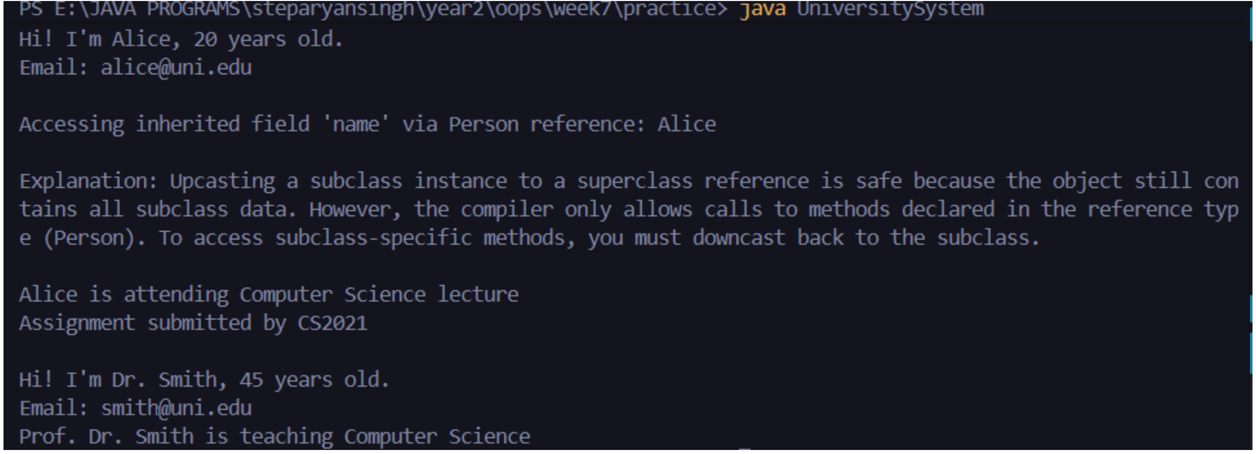
```
PS E:\JAVA PROGRAMS\steparyansingh\year2> cd "e:\JAVA PROGRAMS\steparyansingh\year2\oops\wee
k7\practice"; javac FoodDeliveryDemo.java
PS E:\JAVA PROGRAMS\steparyansingh\year2\oops\week7\practice> cd "e:\JAVA PROGRAMS\steparyan
singh\year2\oops\week7\practice"; java FoodDeliveryDemo
Food Delivery Dynamic Dispatch Demo

Mario's Pizza is making delicious pizza with fresh toppings!
Pizza ready in 20 minutes!

Tokyo Sushi is crafting fresh sushi with precision!
Sushi will be ready in 25 minutes!

Explanation: The JVM uses the actual object's class (runtime type) to resolve overridden met
hods. Even though the reference type is 'Restaurant', the runtime object is PizzaPlace or Su
shiBar, so their overridden methods are invoked.
```



```
public class UniversitySystem {
    public static void main(String[] args) {
        System.out.println("University System Upcasting Demo\n");

        Student alice = new Student("Alice", 20, "alice@uni.edu", 
"CS2021", "Computer Science");

        Person p = alice;
        p.introduce();
        p.getContactInfo();

        System.out.println();

        System.out.println("Accessing inherited field 'name' via Person
reference: " + p.name);

        System.out.println();
        System.out.println("Explanation: Upcasting a subclass instance to
a superclass reference is safe because the object still contains all
subclass data. However, the compiler only allows calls to methods declared
in the reference type (Person). To access subclass-specific methods, you
must downcast back to the subclass.");

        System.out.println();

        if (p instanceof Student) {
            Student s = (Student) p;
            s.attendLecture();
            s.submitAssignment();
        }

        System.out.println();

        Person profRef = new Professor("Dr. Smith", 45, "smith@uni.edu",
"Computer Science");
        profRef.introduce();
        profRef.getContactInfo();

        if (profRef instanceof Professor) {
            ((Professor) profRef).conductClass();
        }
    }
}

class Person {
    protected String name;
    protected int age;
```



```java
    protected String email;

    public Person(String name, int age, String email) {
        this.name = name;
        this.age = age;
        this.email = email;
    }

    public void introduce() {
        System.out.println("Hi! I'm " + name + ", " + age + " years old.");
    }

    public void getContactInfo() {
        System.out.println("Email: " + email);
    }
}

class Student extends Person {
    private String studentId;
    private String major;

    public Student(String name, int age, String email, String studentId, String major) {
        super(name, age, email);
        this.studentId = studentId;
        this.major = major;
    }

    public void attendLecture() {
        System.out.println(name + " is attending " + major + " lecture");
    }

    public void submitAssignment() {
        System.out.println("Assignment submitted by " + studentId);
    }
}

class Professor extends Person {
    private String department;
    public Professor(String name, int age, String email, String
department) {
        super(name, age, email);
        this.department = department;
    }

    public void conductClass() {
        System.out.println("Prof. " + name + " is teaching " +
department);
    }
}
```



```
PS E:\JAVA PROGRAMS\stepary ansingh\year2\oops\week7\practice> java UniversitySystem
Hi! I'm Alice, 20 years old.
Email: alice@uni.edu

Accessing inherited field 'name' via Person reference: Alice

Explanation: Upcasting a subclass instance to a superclass reference is safe because the object still con
tains all subclass data. However, the compiler only allows calls to methods declared in the reference typ
e (Person). To access subclass-specific methods, you must downcast back to the subclass.

Alice is attending Computer Science lecture
Assignment submitted by CS2021

Hi! I'm Dr. Smith, 45 years old.
Email: smith@uni.edu
Prof. Dr. Smith is teaching Computer Science
```