**Week 7 - S7 - Core OOP - Polymorphism - Practice Problem**

Name:Ramesh Harisabapathi Chettiar

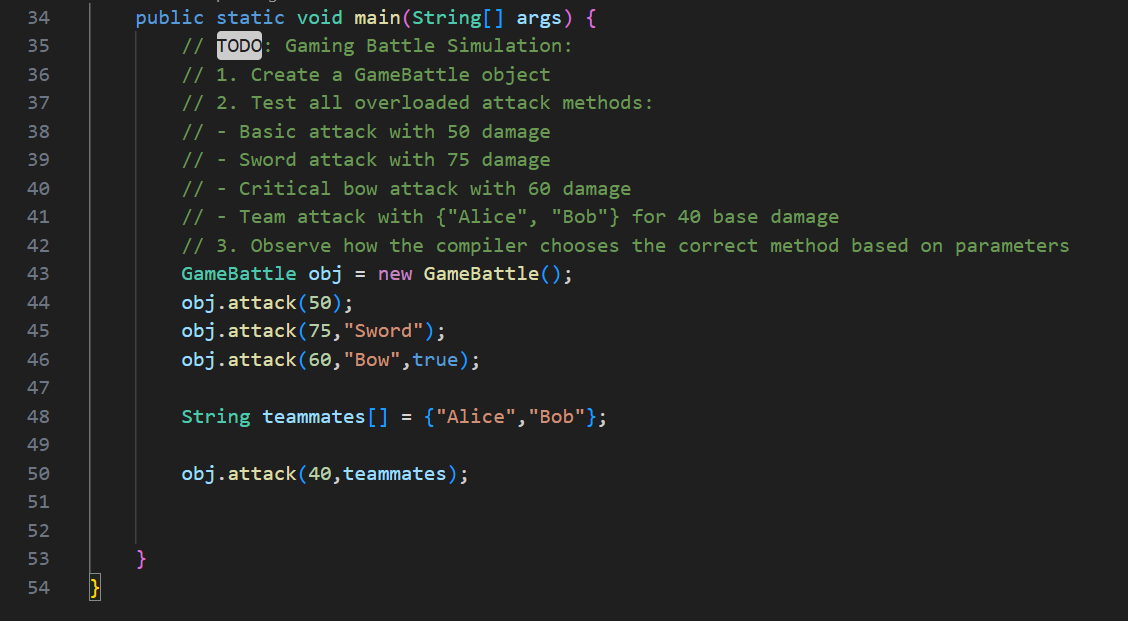
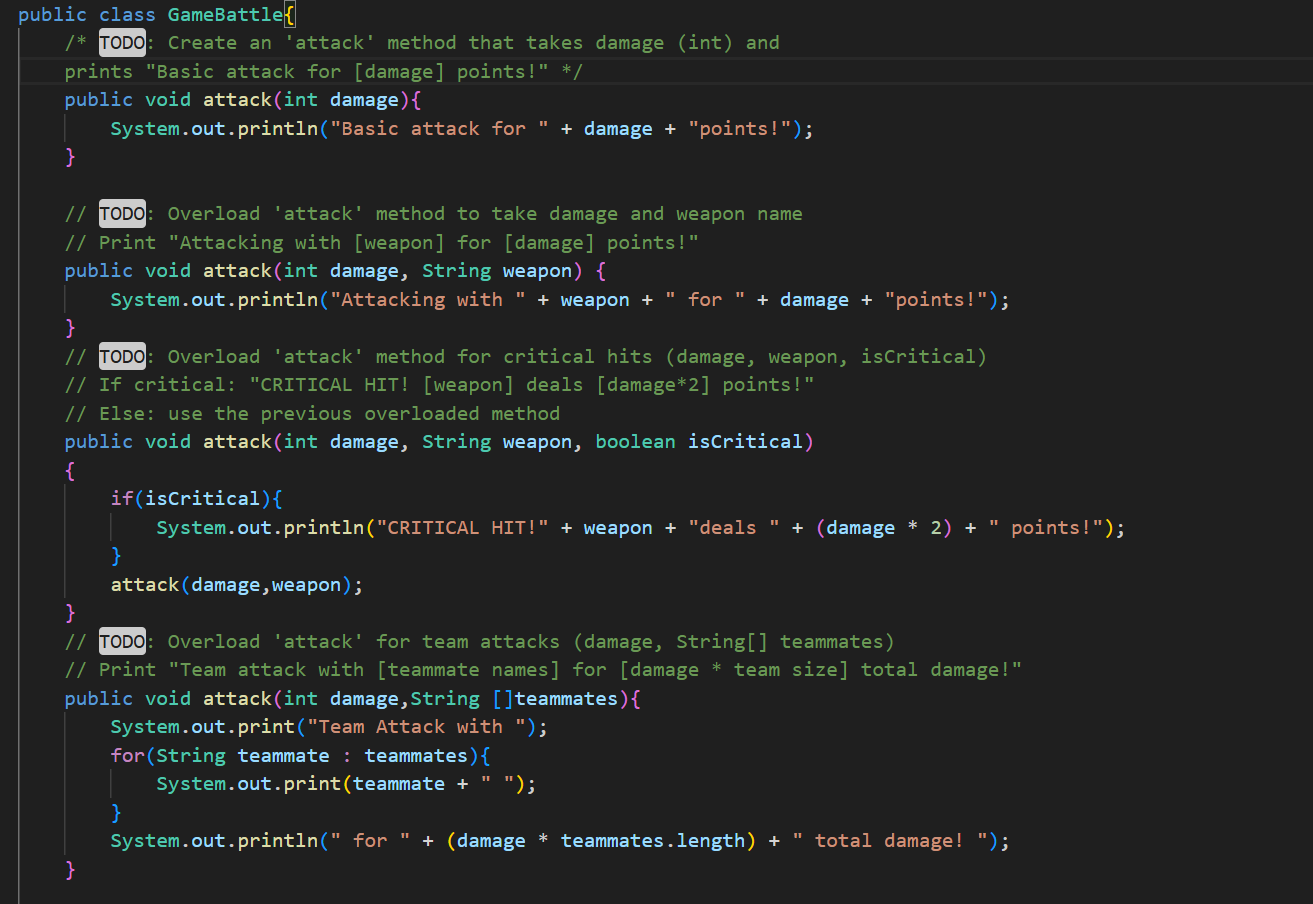
Submitted on:23/09/25

Qno1->

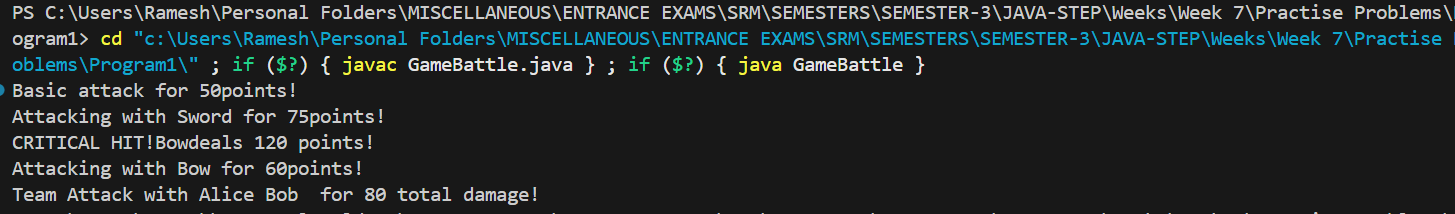
Understanding compile-time polymorphism through method overloading in a

gaming context.

**PROGRAM🡪**

****

**OUTPUT🡪**

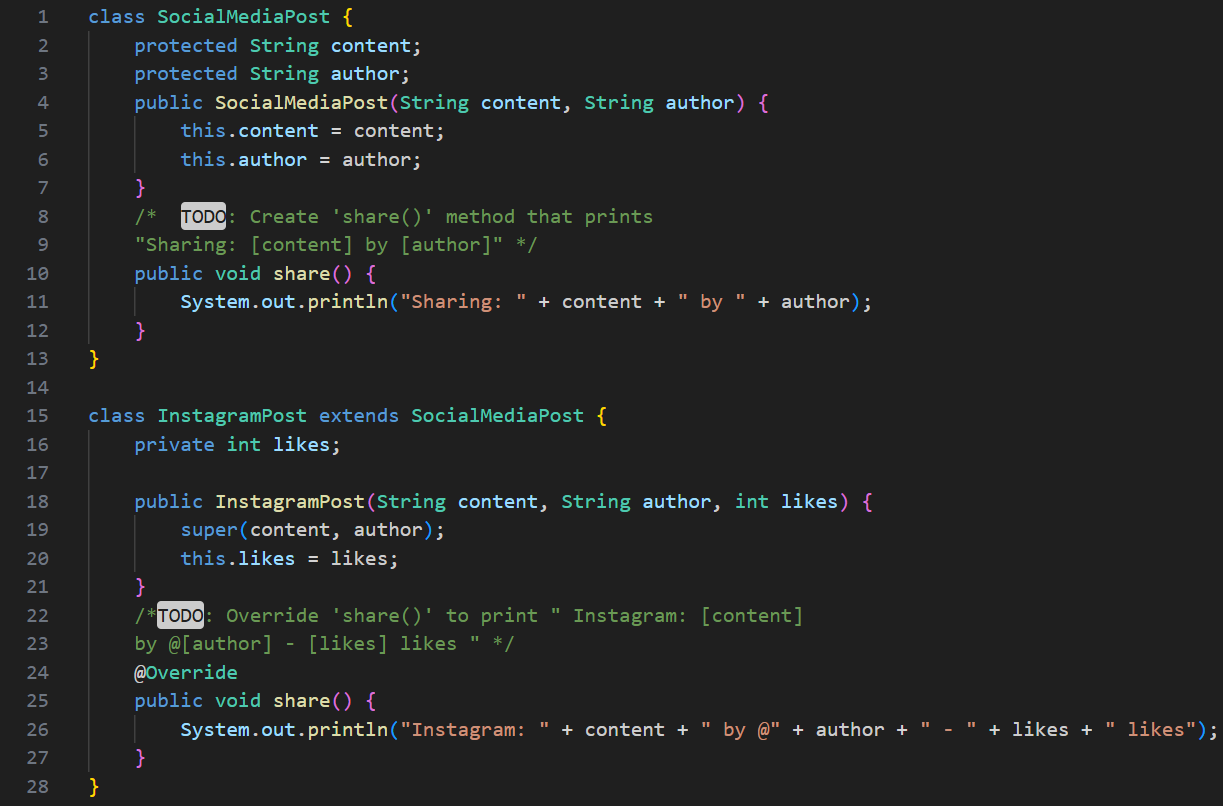
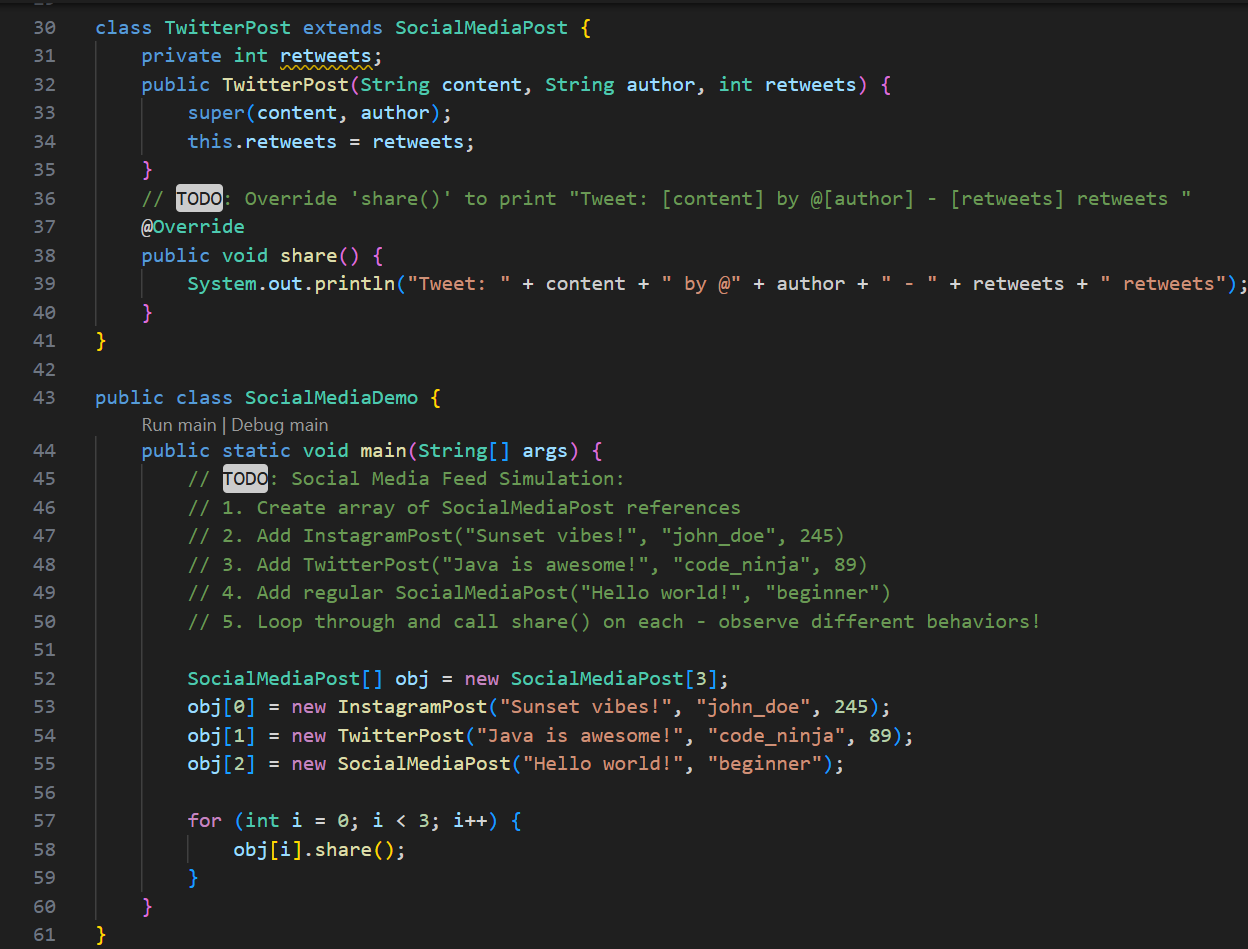
****

Qno2->

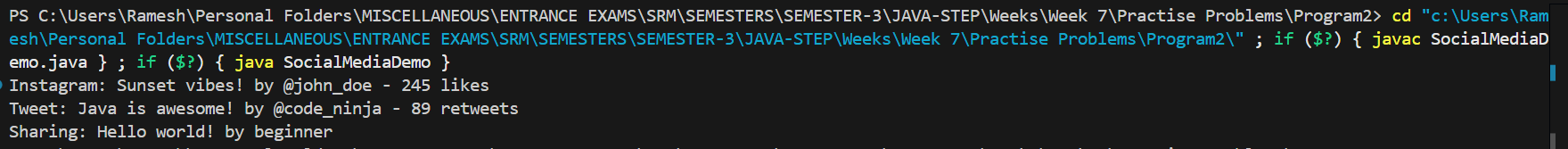
Demonstrating runtime polymorphism through method overriding in social media

context.

**PROGRAM🡪**

**** ****

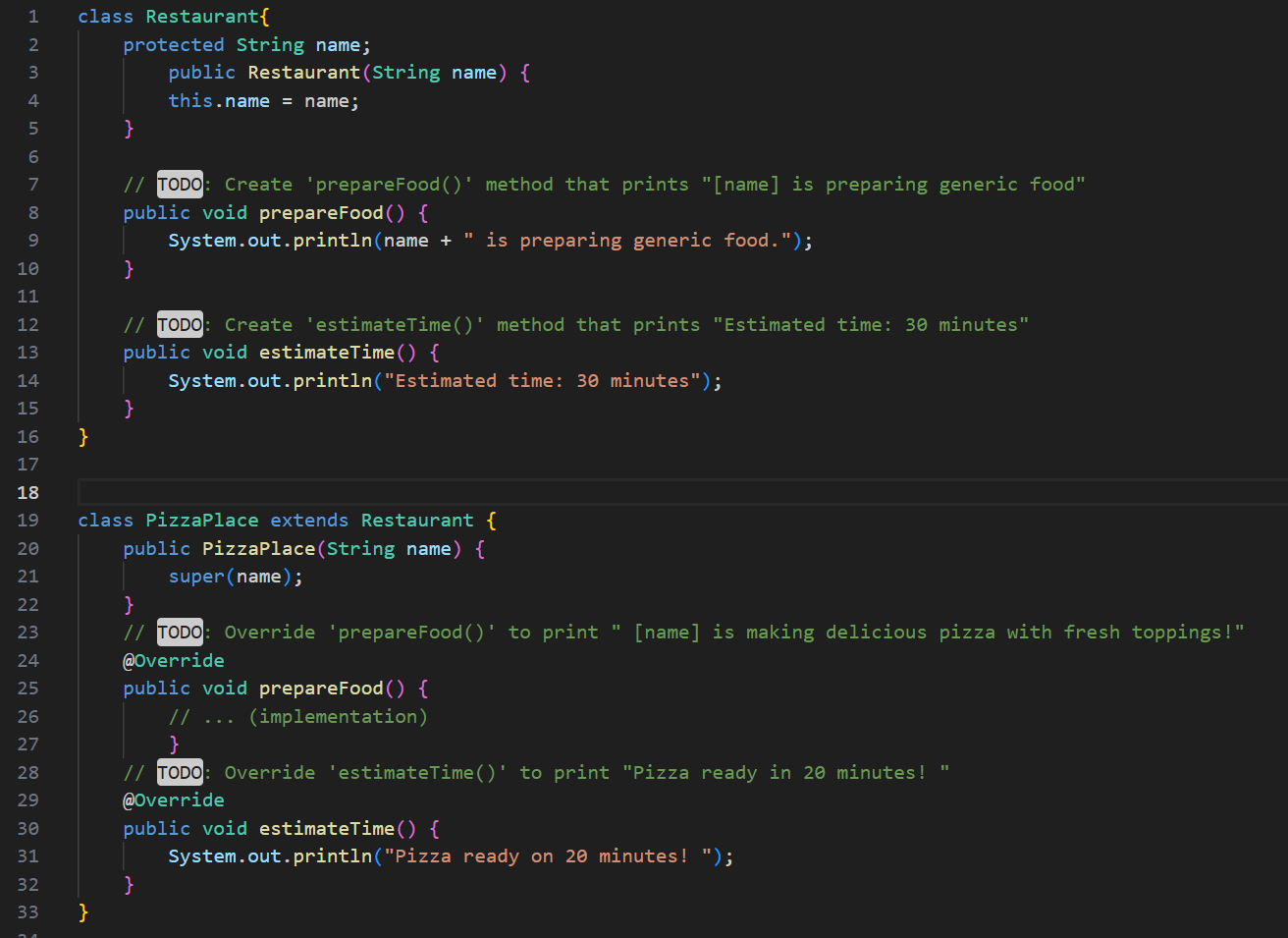
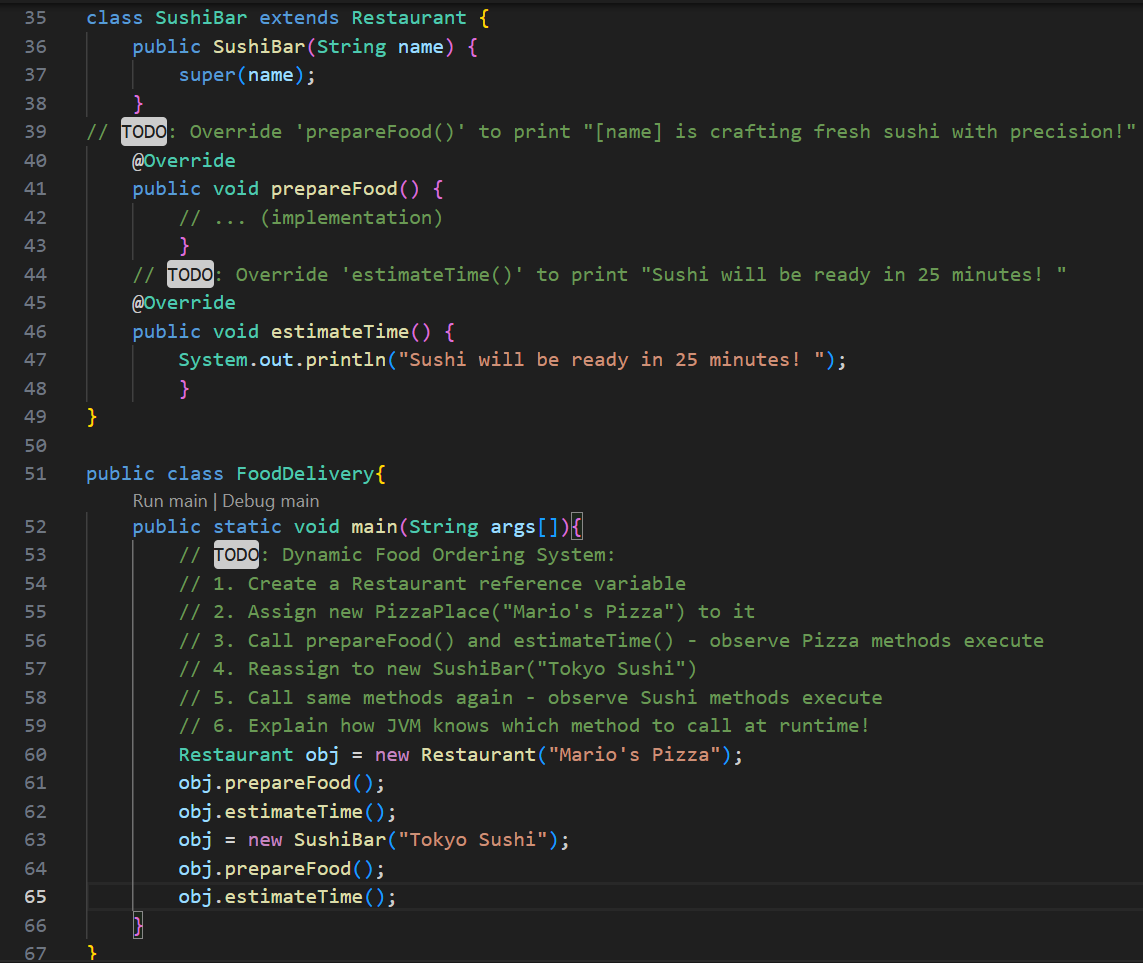
**OUTPUT🡪**

****

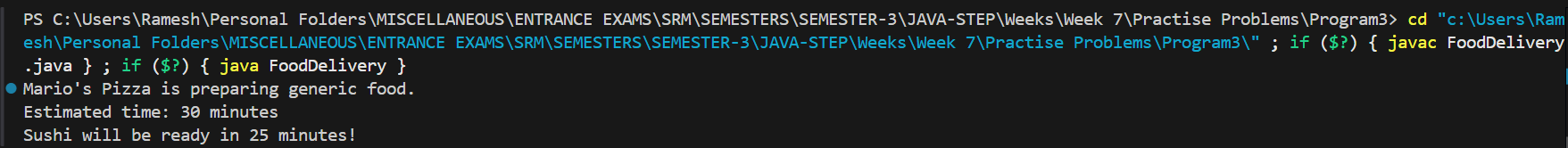
**QNO 3🡪**

**Exploring how JVM resolves method calls at runtime based on actual object type.**

**PROGRAM🡪**

**** ****

**OUTPUT🡪**

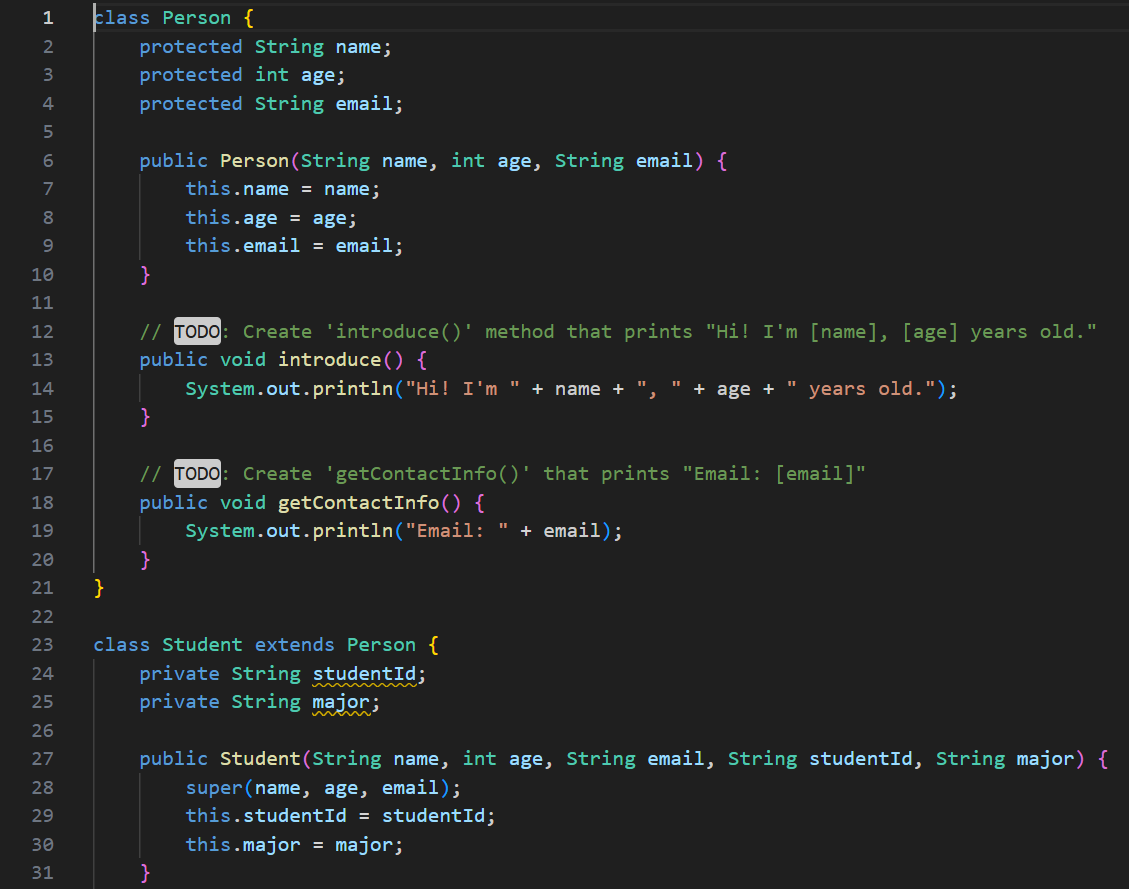
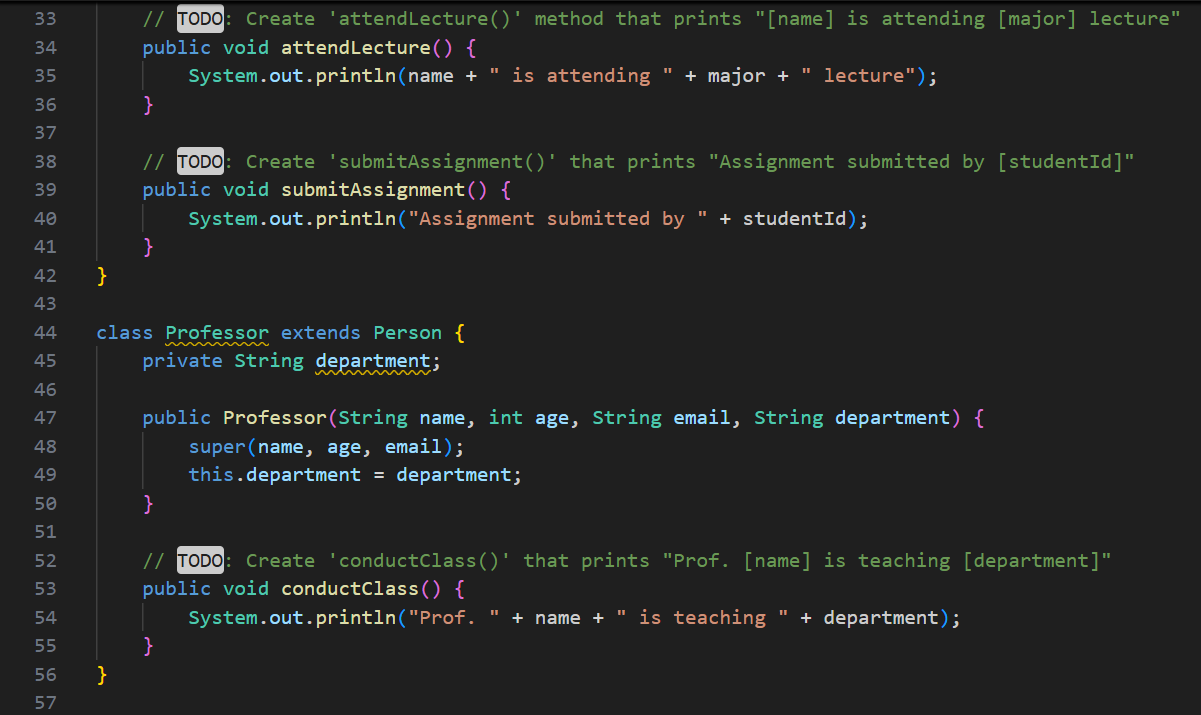
****

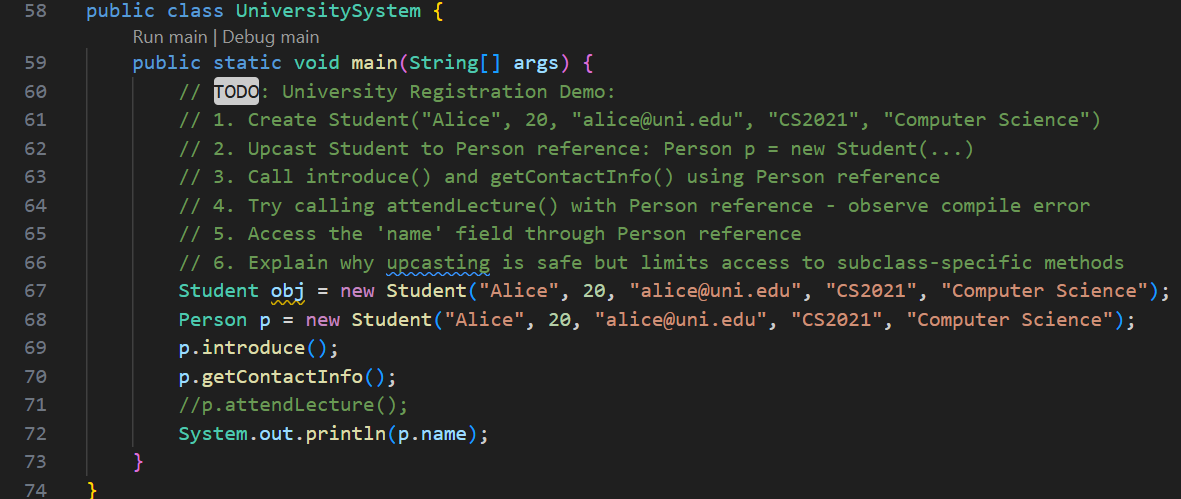
**The JVM uses dynamic method dispatch to determine which method to call at runtime. It relies on the actual object's type, not the reference type, to invoke the correct overridden method—this enables polymorphism. So even if you upcast, the JVM still calls the subclass's version of the method.**

**QNO 4🡪**

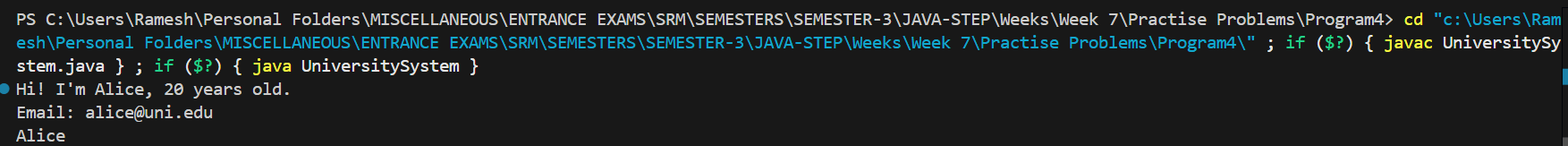
**Learning safe upcasting and accessing inherited members in university context.**

**PROGRAM🡪**

**** ****

****

**OUTPUT🡪**

****

**Upcasting is safe because a subclass object always "is-a" superclass object, preserving type compatibility. However, it limits access to subclass-specific methods since the reference type only exposes the superclass's interface. You can regain access via downcasting if needed.**