The League of Lazy Innovators

Estimated time: 30 minutes
Learning Goals: Practice writing Java-style pseudocode that shows abstract classes, interfaces, polymorphism, and appropriate use of this through designing a chaotic and funny system.

Story Background

Welcome to the League of Lazy Innovators! We are a secret society who dedicate their entire soul and existence into one single thing: doing as little as possible while getting everything done. I mean the name gives it away.

You guys are super genius and brilliant but at the same time, extremely lazy. To avoid work, you usually dream of futuristic systems to automate mundane tasks such as:

- Fridges that teleport your favourite snacks
- Self-walking dogs that argue about which path to take
- Coffee machines that insults you until you drink the coffee
- Smart kitchen that cleans every dish, glass, plate and stoves without your command

Your mission is to design the backend system that manages all these futuristic inventions and entities. Make it funny, chaotic, clever but structured enough to make it "work".

Part 1: Choose Your System (5-10 mins)

You may pick a system from below to design or you can invent your own invention as you have free will.

- CoffeeBot: Coffee machines with emotions that can mirror yours and make coffee based on your emotions
- Responsible Doggos: Dogs that walk themselves (and potentially argue)
- Snack Teleporters: Fridges that bring you your favorite snacks
- Procrastination Bot: Robots that do everything for you but at the last minute
- Sleepy Alarm Clock: Keeps snoozing until insulted
- Chatty Oven: Gives unsolicited advice while baking something
- Levitating Backpack: Floats and follows you, but occasionally gets lost
- Mood-Swing Smart Lamp: Changes light colours depending on your mood, or theirs (may refuse to turn on when feeling rebellious)

Part 2: Design in Java-Style Pseudocode (15 mins)

1) Abstract Class (Base Identity)

Represents all inventions in your system.

Example pseudocode:

```
abstract class Invention
     attribute name
     attribute energyLevel

     constructor(name, energyLevel)
          this.name = name
          this.energyLevel = energyLevel

     method describe()
          print name + “ has energy ” + energyLevel

     abstract method performAction()
```

2) Interfaces (abilities)

Example pseudocode:

```
interface Explodable
     method explode()

interface Gossipable
     method gossip()

interface Teleportable
     method teleport()
```

3) **Subclasses - Specific Inventions**

You're required to create THREE subclasses.

Example pseudocode:

```
class CoffeeBot extends Invention implements Explodable,
Gossipable
     constructor(name, energyLevel)
          super(name, energyLevel)

     method performAction()
          print name + " brews espresso only when feeling on
fire!"

     method explode()
          print name + " explodes coffee everywhere when
provoked!"
     method gossip()
          print name + " spills the tea!"
```

Part 3: Polymorphism (5 mins)

Polymorphism means many forms, one action. In Java, it means the same method call can behave differently depending on the object or the parameters.

Runtime Polymorphism	**Compile-Time Polymorphism**
Method Overriding	Method Overloading
Same method but different classes	Same method but different parameters
Decided at runtime	Decided at compile time

```
abstract class Animal
     abstract method makeSound()

class Dog extends Animal
     method makeSound()
          print "Woof!"
```

```
class Cat extends Animal
     method makeSound()
          print “Meow”

Animal newPet ← new Dog()
call newPet.makeSound( )
```

1) Which version of makeSound( ) runs?
2) When is that decision made? Is it compile time or run time?
3) Why?

Now, apply the same concept to your system by writing:

For example,

```
Invention device ← new CoffeeBot()
call device.performAction()
```

Explain which version runs. Why?