Digital Swiss Army Knife - A Smartphone?

A futuristic tech company is developing a **modular smartphone** that behaves like a digital Swiss Army knife.
The device must:

- store **state that changes over time** (like battery and owner),
- gain **multiple independent abilities** (camera, music, etc.),
- and allow **apps to interact with it polymorphically** without knowing the exact device type.

However, Java has a limitation: a class can inherit from **only one parent class**.

Your job is to design the architecture so the smartphone can still become a **multi-purpose tool** while respecting Java's rules.

The Shared Core

All smart devices in the company share:

- a **battery that drains and recharges**
- an **owner assigned after manufacturing**
- behavior that should **not be duplicated** in every device class

Design an **abstract base type** to hold this shared logic.

Requirements

- The battery value must be **modifiable after creation**.
- Charging should allow **method chaining** so future engineers can write fluent code.
- Avoid confusion between **parameters and object state**.

Plug-in Abilities

Different teams build **independent hardware modules**:

- one team builds **photo capability**
- another builds **music playback**

These modules must work with **any future device**, not just smartphones.

Design them so:

- they describe **what a device can do.**
- but **not how it is implemented**,
- and allow a single class to adopt **many abilities at once**.

Building the SmartPhone

Now assemble the real product.

The smartphone must:

- reuse the **shared core**
- adopt **both plug-in abilities**
- react to user actions in a believable way
  (for example, actions should affect battery life)

The App Store Test

External developers will interact with devices **without knowing the exact model**.

Demonstrate:

1. **Fluent setup** of a smartphone in one expression.
2. **Runtime decision-making** by Java when a generic reference points to a smartphone.

Test: Use the provided Test.java file to test your code.

Code Skeleton

```
// CORE DEVICE BLUEPRINT
abstract class Tool {

    protected int batteryLevel;
    private String owner;

    // Recharge the device in a fluent style
    // (Hint: avoid shadowing confusion)
    public Tool charge(int amount) {
        // TODO
        return this;
    }

    // Assign device ownership after purchase
    // Must support chained calls
    public Tool setOwner(String name) {
        // TODO
        return this;
    }

    // A shared behavior available to all tools
    public void showStatus() {
        System.out.println(owner + "'s device → Battery: " +
batteryLevel + "%");
    }
}


// ABILITY CONTRACTS
interface Camera {
    void takePhoto();
}

interface MusicPlayer {
    void playSong();
}
```

```java
// FINAL PRODUCT
class SmartPhone extends Tool implements Camera, MusicPlayer {

    @Override
    public void takePhoto() {
        // TODO: simulate battery usage + message
    }

    @Override
    public void playSong() {
        // TODO: simulate battery usage + message
    }
}


// APP STORE SIMULATION
public class App {
    public static void main(String[] args) {

        // Fluent construction of a ready-to-use phone
        // TODO

        // Runtime polymorphism demonstration
        // A generic tool reference pointing to a SmartPhone
        // TODO

        // Call shared behavior through the generic reference
        // TODO
    }
}
```