# Model 1 Java Interfaces

An interface is similar to an abstract class, except that all methods are automatically `public` and `abstract`. Likewise, all fields are automatically `public`, `static`, and `final`. These keywords are omitted in the interface definition.

```
public interface Rechargeable {
    int MAX_CHARGE = 10;

    int getCharge();

    void recharge();
}
```

Classes do not extend interfaces; they implement them:

```
public class CellPhone implements Rechargeable {
    private int chargeLevel;
    private int volume;

    public CellPhone(int chargeLevel, int volume) {
        this.chargeLevel = chargeLevel;
        this.volume = volume;
    }

    public int getCharge() {
        return chargeLevel;
    }

    public void recharge() {
        chargeLevel = MAX_CHARGE;
    }

    public int getVolume() {
        return volume;
    }

    public void setVolume(int volume) {
        this.volume = volume;
    }

    public void makeCall() {
        System.out.println("Ring... Hello?");
    }
}
```

## Questions (15 min) Start time:

**1**. What two methods are required by `Rechargeable`?

**2**. Modify your *ToyRobot.java* to implement the `Rechargeable` interface. What changes did you need to make?

**3**. Consider the following `rechargeAll` method. What type of objects are stored in the list?

```
public static void rechargeAll(ArrayList<Rechargeable> list) {
    for (Rechargeable item : list) {
        item.recharge();
    }
}
```

**4**. Consider the following `main` method. Explain the significance of storing `ToyRobot` and `CellPhone` objects in the same `ArrayList` when calling `rechargeAll`.

```
public static void main(String[] args) {
    ArrayList<Rechargeable> items = new ArrayList<>();
    items.add(new ToyRobot());
    items.add(new CellPhone(4, 5));
    rechargeAll(items);
}
```

**5**. Explain how an interface is like a contract.