JAVA OOP BASICS PART II

AGENDA

- **Inheritance**
- **Polymorphism**
- **Overloading**
- **Overriding**
- **Encapsulation**
- Abstraction
- Interfaces and abstraction
- Exceptions

Inheritance

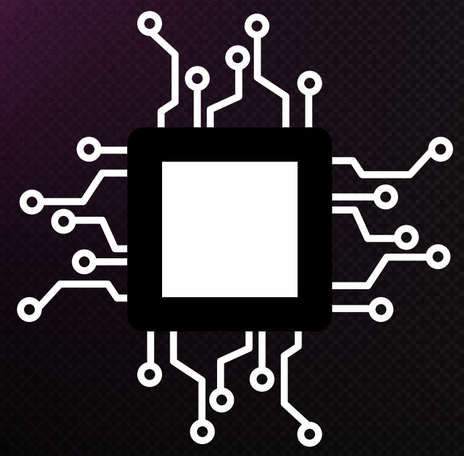
Extending

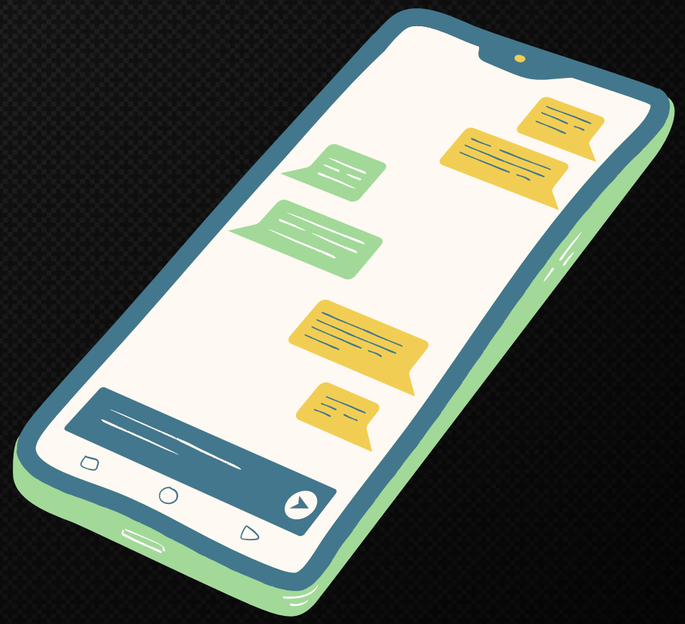
```
public abstract class Electronics extends Product {

    2 usages
    private String voltage;
    2 usages
    private String brand;
```

```
public class Product {
    2 usages
    String name;
    2 usages
    double price;
    2 usages
    Category category;
    2 usages
    ArrayList<Review> reviews;
}
```

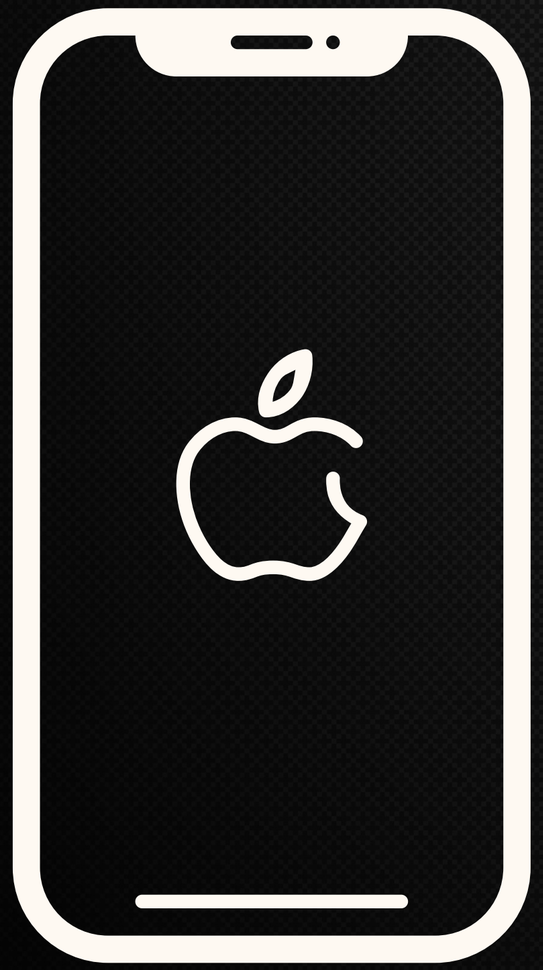
Multiple levels





Polymorphism

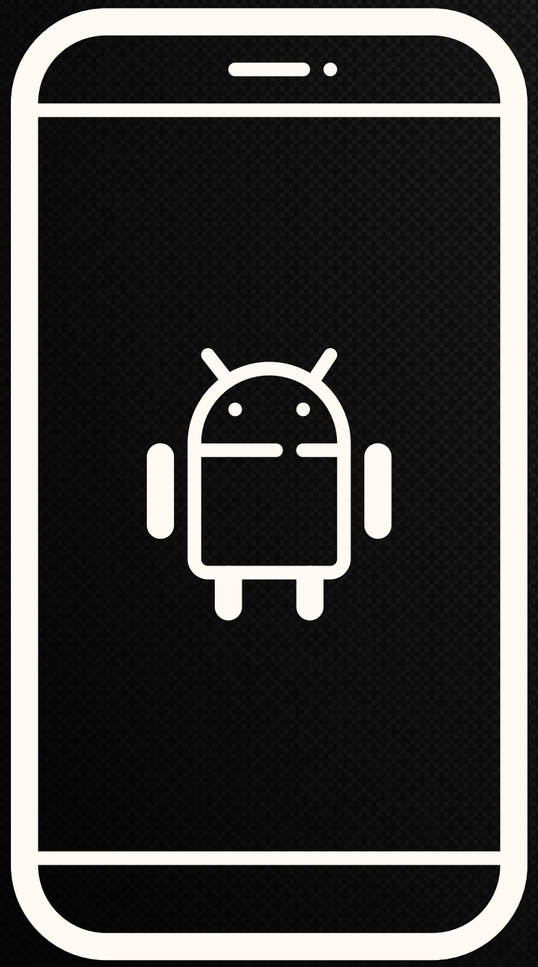
- takeScreenshot()
- restart()
- downlaodApp()



takeSceenshot():
"Press lock and lock and volume up"

restart():
"Press and hold something"

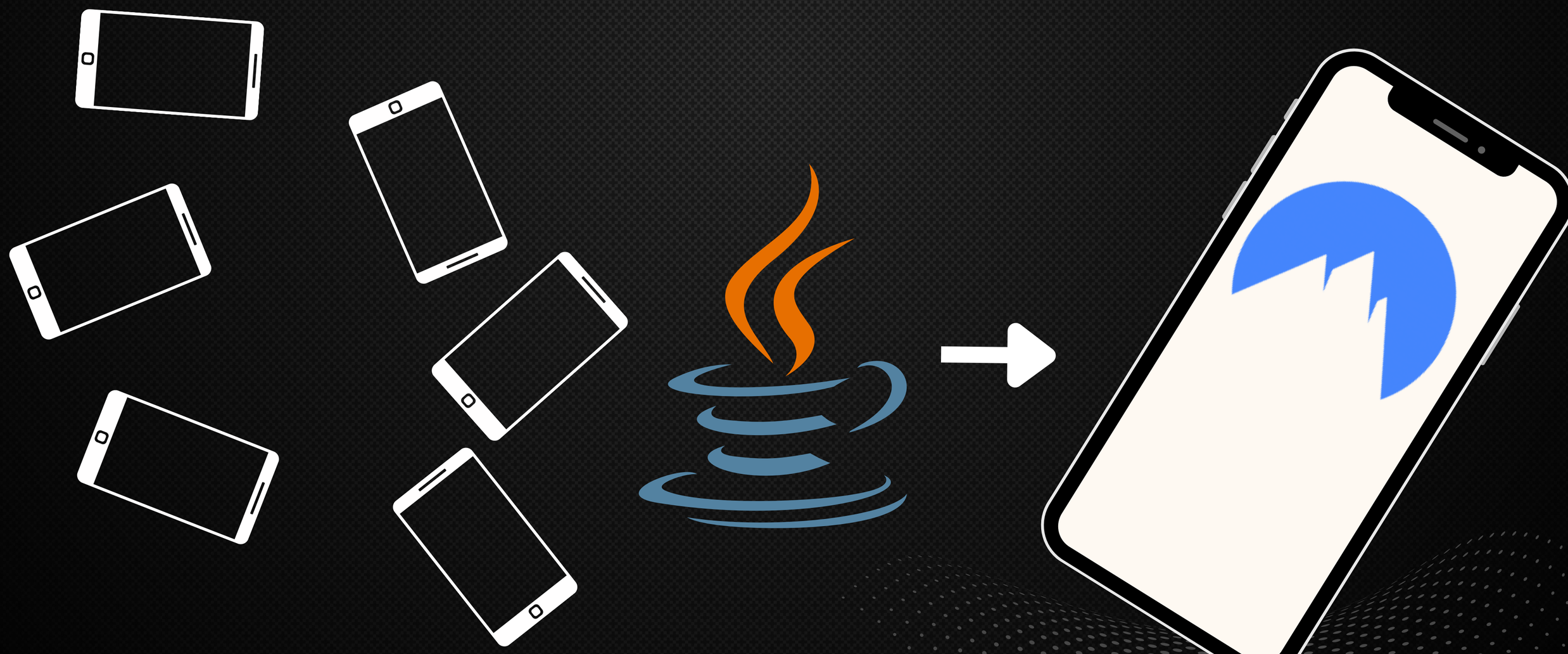
downloadApp():
"go to app store and pay for something"



takeSceenshot():
"Press volume up and down"

restart():
"Press and hold lock button"

downloadApp():
"go to Google Play and get an app"



Abstract Class

```
public abstract class Phone {
    no usages  2 implementations  new *
    abstract void downloadApp();

    no usages  2 implementations  new *
    abstract void restart();

    no usages  2 implementations  new *
    abstract void takeScreenshot();
}
```

```java
public class Iphone extends Phone {
    no usages  new *
    @Override
    void downloadApp() {
        System.out.println("Downloading app from App Store");
    }

    no usages  new *
    @Override
    void restart() {
        System.out.println("Restarting Iphone");
    }

    no usages  new *
    @Override
    void takeScreenshot() {
        System.out.println("Taking screenshot");
    }
}
```

```java
public class AndroidPhone extends Phone {
    no usages  new *
    @Override
    void downloadApp() {
        System.out.println("Downloading app from Google Play"
    }

    no usages  new *
    @Override
    void restart() {
        System.out.println("Restarting Android Phone");
    }

    no usages  new *
    @Override
    void takeScreenshot() {
        System.out.println("Taking screenshot");
    }
}
```

Interface

```java
public interface Downloadable {
    no usages  2 implementations  new *
    void downloadApplication();
}
```

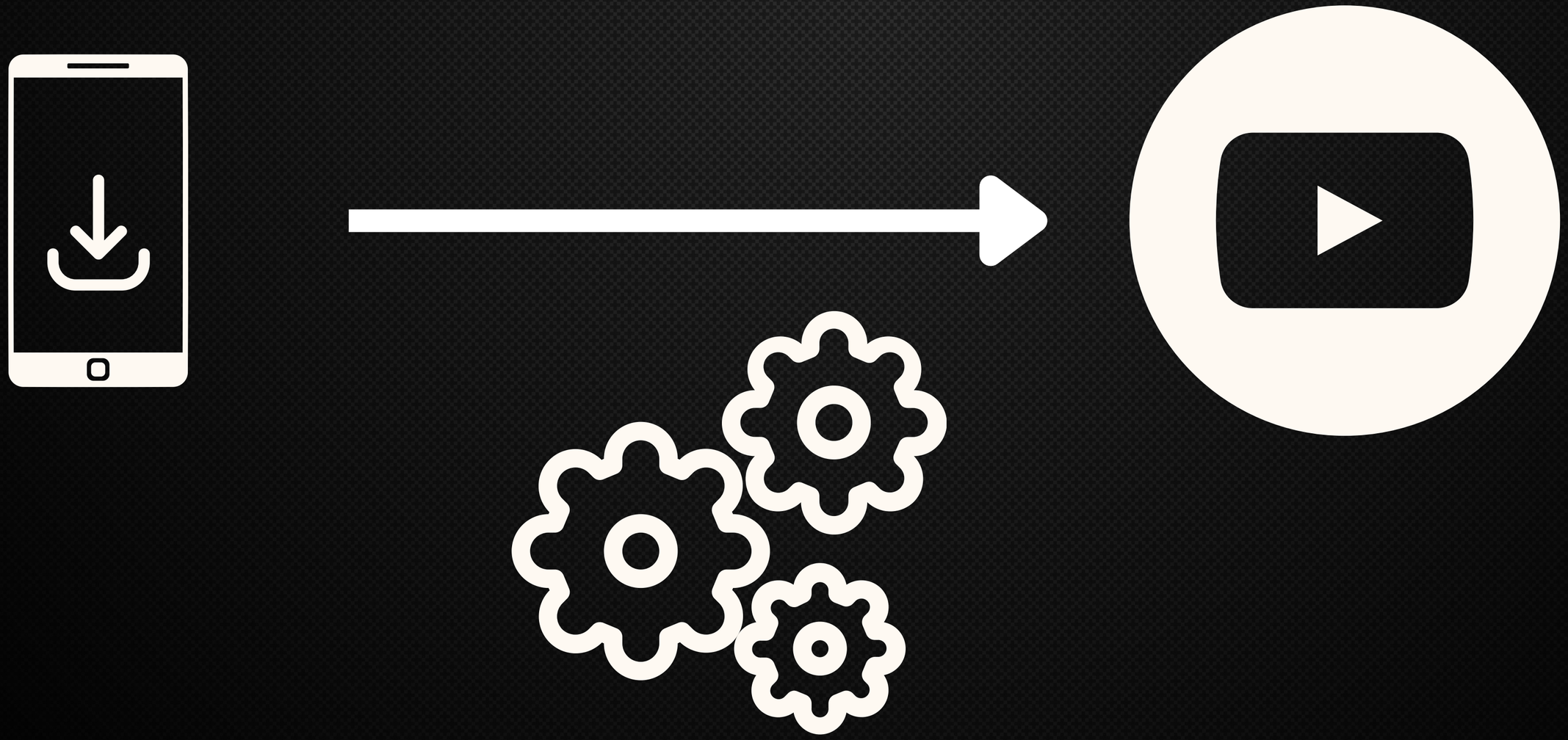
```java
public class AndroidPhone implements Downloadable {
    no usages  new *
    @Override
    public void downloadApplication() {
        System.out.println("Download application from Google Play");
    }
}
```

```java
public class Iphone implements Downloadable {
    no usages  new *
    @Override
    public void downloadApplication() {
        System.out.println("Download application from AppStore");
    }

}
```

```java
public void downloadApps(Downloadable downloadable) {
    downloadable.downloadApplication();
}
```

Encapsulation



Exceptions

```java
public double getAverageRating() {
    double sum = 0;
    for (Review review : reviews) {
        sum += review.getRating();
    }
    return sum / reviews.size();
}
```

```java
public String getName() { return name; }

3 usages   1 override   ZilvinasVysniauskas
public double getPrice() { return price;
```

```java
public void setPrice(int price) { this.price = price; }
```

Encapsulation

```java
public double getAverageRating() {
    double sum = 0;
    for (Review review : reviews) {
        sum += review.getRating();
    }
    return sum / reviews.size();
}
```

```java
public String getName() { return name; }

3 usages   1 override   ZilvinasVysniauskas
public double getPrice() { return price;
```

```java
public void setPrice(int price) { this.price = price; }
```

Exceptions

Checked exceptions

↓

Compile

↓

ClassNotFoundException
FileNotFoundException
SQLException

Unchecked exceptions

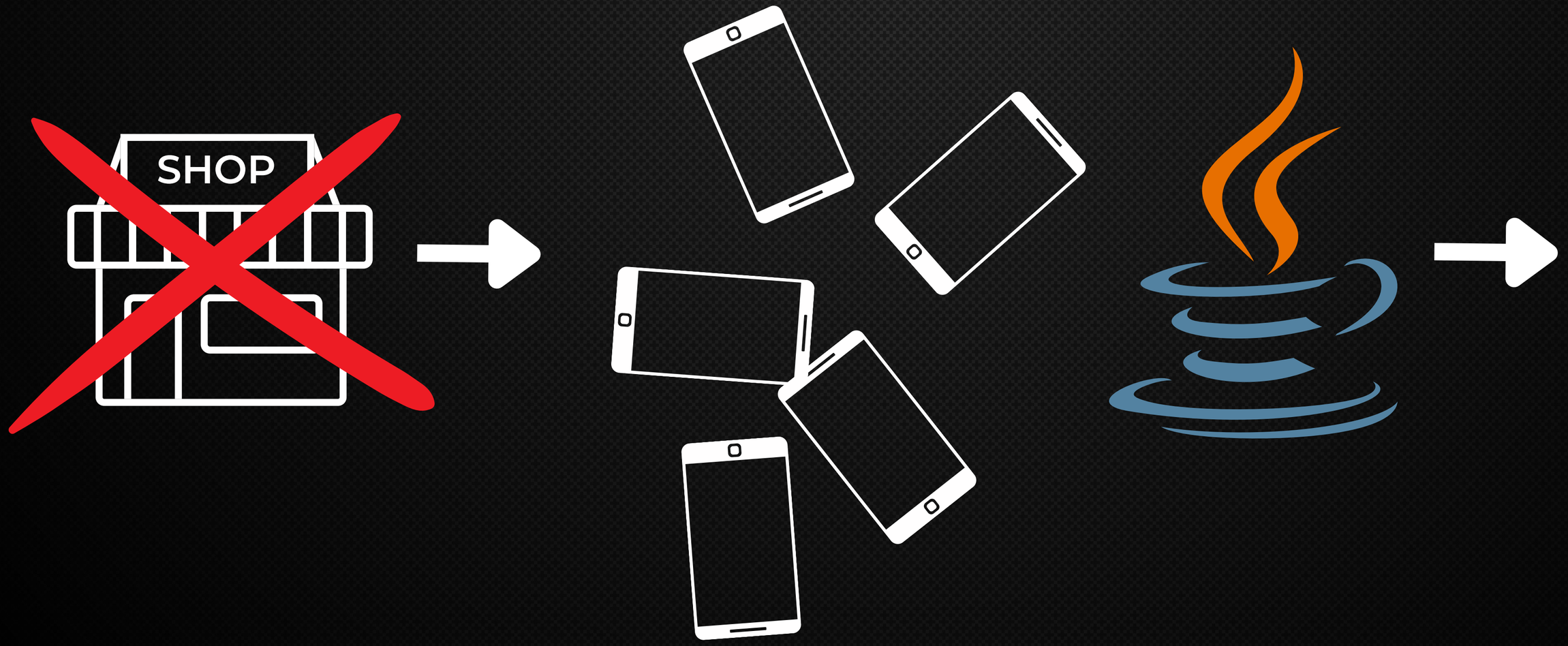
↓

Compile

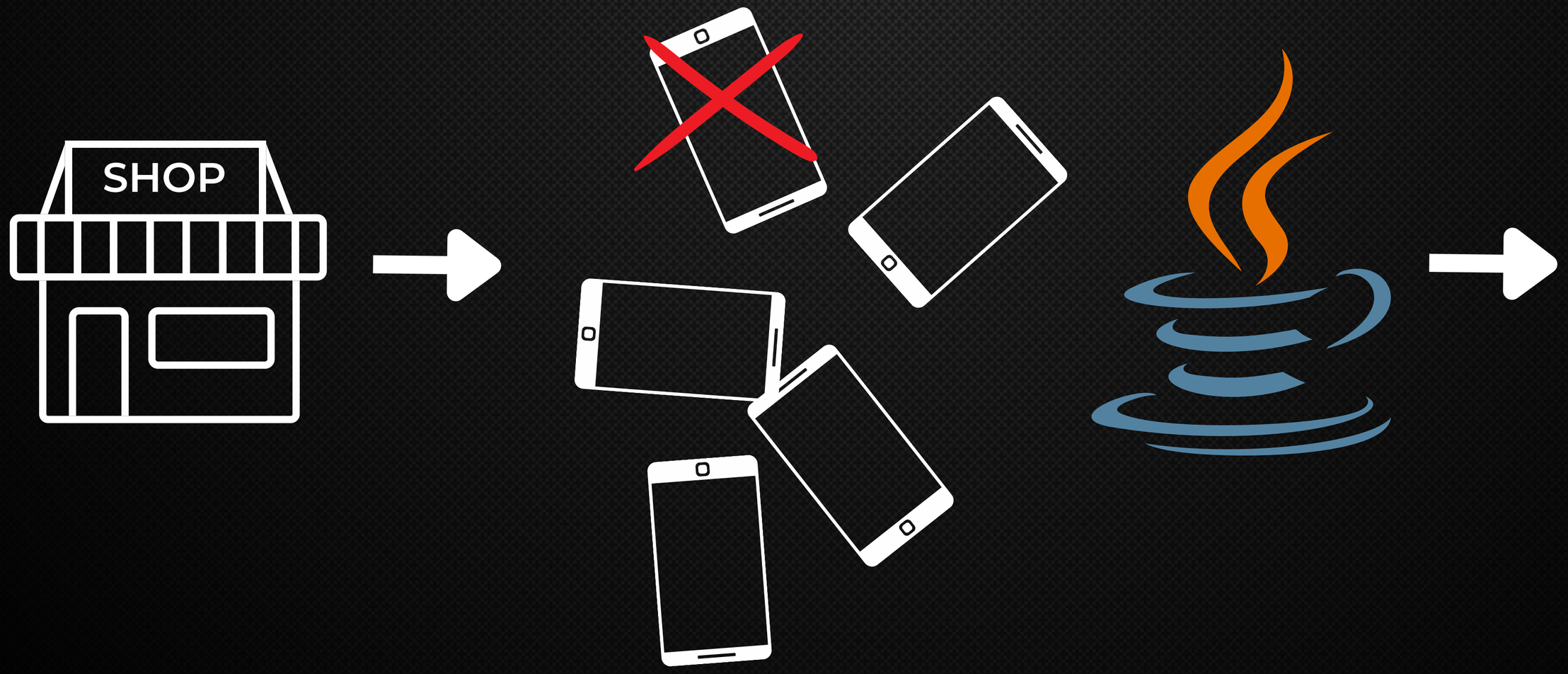
↓

NullPointerException
ArrayOutOfBoundException
ClassCastException

Checked ClassNotFoundException



Unchecked NullPointerException



Custom Exceptions

```
public class ProductNotFoundException extends Exception {
    3 usages   ZilvinasVysniauskas
    public ProductNotFoundException(String message) {
        super(message);
    }

}
```

Throwing Exception

```java
public Product getProductByName(String name) throws ProductNotFoundException {
    for (Product product : products) {
        if (product.getName().equals(name)) {
            return product;
        }
    }
    throw new ProductNotFoundException("Product " + name + " was not found");
}
```

Handling

```java
private void leaveReviewsAndRatingsForProducts() {
    /**
     * User leaves review for each product in the shopping cart.
     */
    Review review = new Review( author: "John Doe",  rating: 5,  comment: "This is a review for the shirt");

    try {
        shoppingCart.getProductByName("Book").addReview(review);
    } catch (ProductNotFoundException e) {
        System.out.println("Review was not left, " + e.getMessage());
    }

}
```

Lets write some code...