Tutorial for polymorphism

In this tutorial, you'll continue to build a bookstore application, using polymorphism to expand the store into an unrelated business.

Your bookstore application has allowed the store to expand its product line and achieve great success. The bookstore is now selling books and movies, and has plans to sell more media in the future.

With great success comes more great ideas. The owners have informed you that the stores will open cafes to sell coffee, tea, and other food and drink items. You need to modify your application to handle the purchase of food and drinks in addition to books and movies.

To get started, import this project into IntelliJ.

Design

You could add food and drink items as child classes of `MediaItem`. However coffee, tea, and pastries don't pass the *is a* test as media items. A scone *is NOT a* media item. So class inheritance isn't a good choice here.

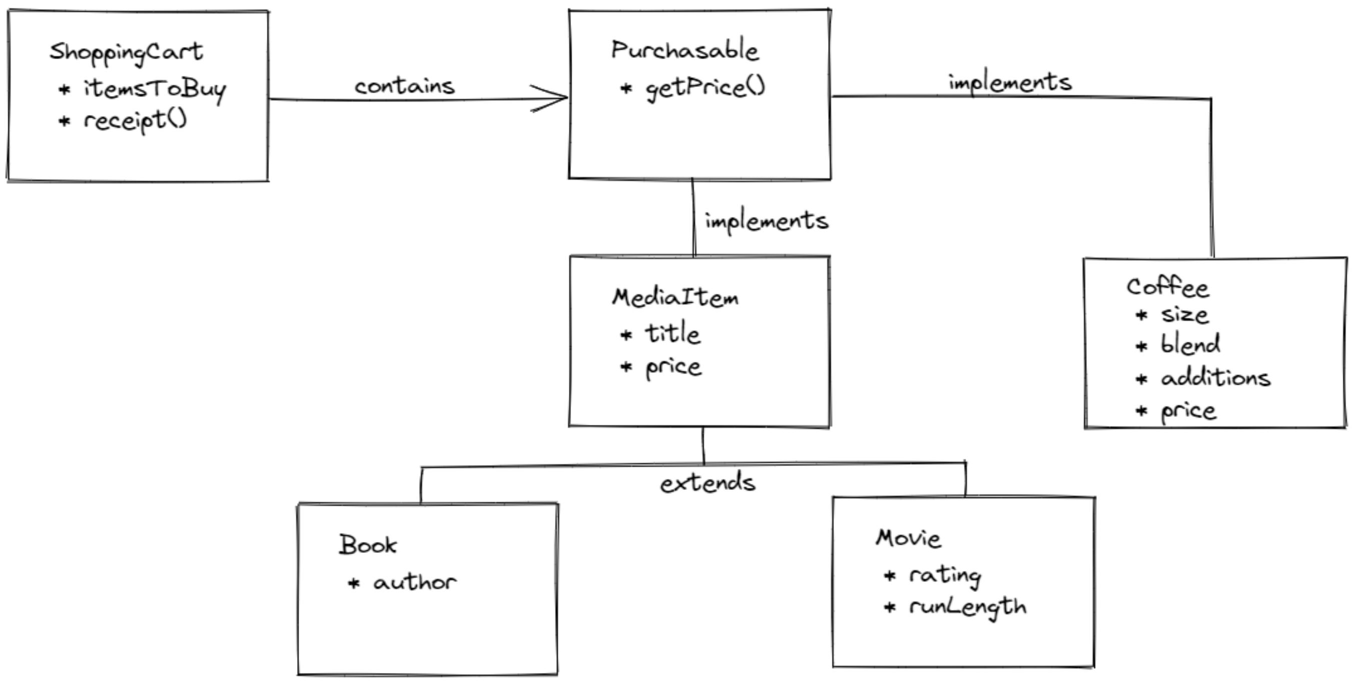
Instead, you define what it takes to make something *purchasable*. For this store, to be purchasable an item must have a price.

You're going to define an **interface** called `Purchasable`. An interface defines a "contract" of properties and methods that any class implementing the interface *must* define.

The `Purchasable` interface you write has one method: `getPrice()`. You store can sell any item that implements the "contract" established by the `IPurchasable` interface.

Your shopping cart must now contain "purchasables"; that is, any class that implements the `Purchasable` interface.

The new class hierarchy looks like:



In the diagram, `Coffee` is the only cafe item defined, but you can add many others, as long as those classes implement the `Purchasable` interface.

Step One: Support the `Purchasable` interface

The first thing to do is *refactor* your existing code to support your new design. Recall that refactoring re-structures your code without changing its function.

Create the `Purchasable` interface

In the IntelliJ Project window, right-click on the `com.techelevator` package and select **New > Java Class** from the menu. Type the name `Purchasable`, and select `Interface` from the list, then press **Enter**. IntelliJ creates the new file `Purchasable.java`, and declares the interface inside it.

Add the following method declarations to `Purchasable`:

```
// Purchasable.java
public interface Purchasable {
    double getPrice();
}
```

Notice that the `getPrice()` method *has no body*. Interfaces define the method *signatures*, but don't include the *implementation* (body) of the method. That's the job of the class which implements the interface.

Implement `Purchasable` in `MediaItem`

A media item (a book or a movie) is purchasable, so implement the new interface in the `MediaItem` class. In `MediaItem.java`, change the class declaration:

```
// MediaItem.java
public class MediaItem implements Purchasable {
    ...
```

Since `MediaItem` already has a `getPrice()` method, there's nothing else you need to do. `MediaItem` now completely implements the `Purchasable` interface.

Change `ShoppingCart` to hold "purchasables"

Currently the shopping cart contains a list of media items to purchase. Change that to a list of `Purchasable`. This means the shopping cart holds a *list of items that are purchasable*.

Change the declaration of `itemsToBuy`, as well as the places in the code that refer to the list:

```
// ShoppingCart.java

// *** Change from MediaItem to Purchasable
private List<Purchasable> itemsToBuy = new ArrayList<>();

// *** Change from MediaItem to Purchasable
public void add(Purchasable itemToAdd) {
    itemsToBuy.add(itemToAdd);
}

public double getTotalPrice() {
    double total = 0.0;
    // *** Change from MediaItem to Purchasable
    for (Purchasable item : itemsToBuy) {
        total += item.getPrice();
    }
    return total;
}

public String receipt() {
    String receipt = "\nReceipt\n";
    // *** Change from MediaItem to Purchasable
    for (Purchasable item : itemsToBuy) {
        receipt += item;
        receipt += "\n";
    }

    receipt += "\nTotal: $" + getTotalPrice();

    return receipt;
}
```

When you run the program now, you get the same output as you had before making these changes. You've refactored the code so that the shopping cart can hold any kind of purchasable item:

```
Welcome to the Tech Elevator Bookstore

Receipt
Title: A Tale of Two Cities, Author: Charles Dickens, Price: $14.99
Title: The Three Musketeers, Author: Alexandre Dumas, Price: $12.95
Title: Childhood's End, Author: Arthur C. Clark, Price: $5.99
Title: Toy Story, Rating: G, Time: 81 minutes, Price: $19.99
Title: Airplane!, Rating: PG, Time: 88 minutes, Price: $14.99

Total: $68.91
```

Step Two: Define a purchasable cafe item

Now create your first cafe item to sell. You can create many types of cafe items for purchase, but start with `Coffee`.

In the IntelliJ Project window, right-click on the `com.techelevator` package and select **New > Java Class**. Type the name of the class, `Coffee` and press **Enter**.

Change the class declaration to indicate that `Coffee` implements `Purchasable`:

```java
// Coffee.java
public class Coffee implements Purchasable {
```

In the class, define the properties and methods for `Coffee`: `size`, `blend`, `additions`, and `price`:

```java
// Coffee.java
private String size;
private String blend;
private List<String> additions = new ArrayList<>();
private double price;

public String getSize() {
    return size;
}

public void setSize(String size) {
    this.size = size;
}

public String getBlend() {
    return blend;
}

public void setBlend(String blend) {
    this.blend = blend;
}
```

```
    @Override
    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }

    public void add(String addition) {
        additions.add(addition);
    }
```

> Note: IntelliJ may add `import` statements for the packages you use automatically as you add the properties. If it doesn't, make sure you include these imports before your `Coffee` class declaration:

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
```

Notice that `getPrice()` satisfies the "contract" for the `Purchasable` interface.

For coffee to display well on a receipt, implement the `toString()` override:

```
// Coffee.java
@Override
public String toString() {
    return size + " coffee, " +
            blend + " (" +
            String.join(",", additions) +
            "). Price: $" + price;
}
```

Finally, for convenience, add a constructor to `Coffee`:

```
// Coffee.java
public Coffee(String size, String blend, String[] additions, double price) {
    this.size = size;
    this.blend = blend;
    this.additions.addAll(Arrays.asList(additions));
    this.price = price;
}
```

Step Three: Purchase coffee and books

Now that you've added coffee as a purchasable item, you can add some to your cart to make a purchase. Do this in `Bookstore.java`, after you have added media items to the cart:

```
// Bookstore.java
// Have a cuppa jo!
Coffee myCoffee = new Coffee("Extra-large", "Dark Roast", new String[] {"Creme"},
3.99);
Coffee myFriendsCoffee = new Coffee("Medium", "House Blend", new String[]{"Soy
milk", "Sugar"}, 2.79);
shoppingCart.add(myCoffee);
shoppingCart.add(myFriendsCoffee);

System.out.println(shoppingCart.receipt());
```

When your receipt prints, you see that your program handled these seemingly different types of items in a consistent way, and printed a consolidated receipt.

```
Welcome to the Tech Elevator Bookstore

Receipt
Title: A Tale of Two Cities, Author: Charles Dickens, Price: $14.99
Title: The Three Musketeers, Author: Alexandre Dumas, Price: $12.95
Title: Childhood's End, Author: Arthur C. Clark, Price: $5.99
Title: Toy Story, Rating: G, Time: 81 minutes, Price: $19.99
Title: Airplane!, Rating: PG, Time: 88 minutes, Price: $14.99
Extra-large coffee, Dark Roast (Creme). Price: $3.99
Medium coffee, House Blend (Soy milk,Sugar). Price: $2.79

Total: $75.69
```

> Polymorphism is the principle behind this. The word literally means "many forms", and what you've shown is that there are many forms of purchasable items. Drinks are significantly different than books, but your program was able to treat these items similarly when it came to making a purchase.

Summary

In this tutorial, you learned how to:

- Define an interface that describes some common features that otherwise different objects may have.
- Implement an interface inside a class by implementing the methods defined on that interface.
- Polymorphically use disparate types of objects in a similar way by calling methods of an interface.