SE116 – LAB#6 2024-2025 SPRING

Aim: Abstract classes and Interfaces

In this lab, you will create a **robust package shipping system** using **abstract classes** and **interfaces** in Java. The system will include an abstract superclass named **Package**, multiple **concrete subclasses** such as **StandardPackage**, **ExpressPackage**, and **FragilePackage**, as well as **three interfaces—Refundable**, **Trackable**, and **Insurable**. Finally, you will write a **ShippingSystem** class to manage a list of packages, and a **Main** class to demonstrate the program in action. By following these steps, you will deepen your knowledge of inheritance, polymorphism, abstract class and interface implementation.

Begin by writing the **abstract class** named **Package**, which has the following **private** variables:

1. A `String senderName` to hold the name of the person sending the package.
2. A `String recipientName` to hold the name of the person receiving the package.
3. A `double weight` to specify the weight of the package (in kilograms).
4. A `boolean isDelivered` to indicate whether the package has been delivered or not, which defaults to `false`.
5. A `String destinationCity` and a `String destinationCountry` that store the final location details of the shipment.

The **Package** class provides a **no-argument constructor** that initializes these variables to default values, such as empty strings for names and zero for weight, as well as a **parameterized constructor** that accepts the sender's and recipient's names, the weight, and the destination details. Since **Package** is abstract, it must include at least one **abstract method**, which you will name `calculateShippingCost` and declare to return a `double`. This method will force all concrete subclasses of **Package** to define how their shipping costs are computed. You will also write **concrete methods** in `Package` such as `markDelivered` to set `isDelivered` to `true`, `isDelivered` to return the current state of `isDelivered`, and `printInfo` to display the details about the package including its sender, recipient, weight, and delivery status.

Next, you will create the **Refundable** interface, which requires classes that implement it to define a method named `requestRefund` taking a `String reason` parameter and returning a `boolean`, as well as a method named `getRefundAmount` returning a `double`. The interface will also contain a `default void logRefundRequest(String packageIdentifier)` method that prints out a simple log statement for any refund request. Classes implementing this interface can override the default logging if they have a custom behavior.

After creating the **Refundable** interface, you will define the **Trackable** interface. Any package class implementing this interface must define the methods `getTrackingInfo` which returns a `String` and `updateLocation` which accepts a `String newLocation` to update the package's location. In addition, `Trackable` provides two more methods: `setEstimatedDeliveryTime(String dateTime)` and `getEstimatedDeliveryTime()`, allowing for storing and retrieving an estimated delivery time as a `String`.

You will then write the **Insurable** interface, which represents the ability for a package to carry insurance coverage. This interface declares a `void insurePackage(double insuredValue)` method to set the insurance amount, a `double getInsuredValue()` method to retrieve that coverage, and a `boolean claimInsurance(String claimReason)` method, which implementing classes will define to approve or deny an insurance claim. A default method named `logInsuranceClaim(String packageIdentifier, String reason)` may also be included for simple logging if a class does not override it.

Having defined your abstract class and your interfaces, you will write **three concrete subclasses** of **Package**: **StandardPackage**, **ExpressPackage**, and **FragilePackage**. Each one must **extend** `Package` and **implement** any interfaces it claims to support.

First, in **StandardPackage**, you will implement only the **Trackable** interface. This means StandardPackage must define the methods provided through the interface. It overrides the abstract `calculateShippingCost` method from `Package`. An example cost formula might simply be `(weight * 2.0)`. You also store additional private fields such as a `String shippingType` (defaulting to `"Ground"`), a `String currentLocation` to keep track of the package's location, and a `String estimatedDeliveryTime` for your scheduling logic. In the constructor, you will call the `super` constructor of `Package` to initialize sender name, recipient name, weight, destination city, and destination country. You can also override the `printInfo` method to display the shipping type and location details.

Second, in **ExpressPackage**, you will implement both **Trackable** and **Insurable**. Thus, in addition to defining the abstract `calculateShippingCost` (for example, `(weight * 5.0) + 10.0`), you need to provide logic for `getTrackingInfo`, `updateLocation`, `setEstimatedDeliveryTime`, `getEstimatedDeliveryTime` from **Trackable**, and `insurePackage`, `getInsuredValue`, and `claimInsurance` from **Insurable**. This class also has attributes such as an `int priorityLevel`, a `String currentLocation`, a `String estimatedDeliveryTime`, and a `double insuredValue` to track coverage. The `claimInsurance` method can decide whether to approve or deny the insurance claim based on conditions such as "lost" or "damaged." In the constructor, you will again call the `super` constructor of `Package` for common fields, and initialize any class-specific fields, such as priority.

Lastly, in **FragilePackage**, you will implement **Trackable**, **Insurable**, and **Refundable**, as it represents items needing special care and possibly requiring both insurance and refund logic. Your fields for this class are: `boolean requiresReinforcedBox`, `boolean requiresTemperatureControl`, a `String currentLocation`, a `String estimatedDeliveryTime`, a `double insuredValue`, and a `double refundAmount`. By overriding `requestRefund` from **Refundable**, you can determine under which conditions a refund is granted; for example, if a package is delivered and "damaged," you may allow a specific refund amount. By also overriding `markDelivered` in `FragilePackage`, you can print a special note such as "Handle with care – Fragile item delivered!" after calling `super.markDelivered()`. Additionally, you will override `calculateShippingCost` to implement a fragile-specific formula, such as `(weight * 2.0) + 8.0`.

Once your abstract class and interfaces are established, and your three concrete subclasses are implemented, you will build a **ShippingSystem** class that holds an `ArrayList<Package>` to store any `Package` objects. The ShippingSystem class should include methods to `addPackage`, `removePackage`, `printAllPackages`, and `generateReport`. The `addPackage` method will insert a new package into the list if it is not already present. The `removePackage` method

will remove a specified package if it exists. The `printAllPackages` method will loop over every `Package` in the list and call the package's `printInfo`. The `generateReport` method can then output how many packages are currently stored, how many of them are delivered, and the average shipping cost (determined by calling each package's `calculateShippingCost`).

Finally, write your **Main** class to tie everything together. In this class's `main` method, you should do the following:

1. Create an instance of `ShippingSystem`.
2. Instantiate several package objects, such as two or three `StandardPackage` objects, two `ExpressPackage` objects, and one `FragilePackage`. Call their constructors with realistic data, such as names, destination details, and weights.
3. For any class implementing **Trackable**, call `updateLocation` and `setEstimatedDeliveryTime`. Then retrieve these details using `getTrackingInfo` and `getEstimatedDeliveryTime`.
4. For classes implementing **Insurable**, call `insurePackage` to set a coverage amount, and optionally `claimInsurance` if something happens.
5. For the fragile package, call `requestRefund` with different reasons to see if a refund is approved. Log the refund request if you wish, or check how much the refunded amount is by calling `getRefundAmount`.
6. Add all these packages to the shipping system using `addPackage`.
7. Call `printAllPackages` to display each package's information.
8. Demonstrate the `generateReport` method to summarize the shipping costs, number of delivered vs. in-transit packages, and so on. Optionally mark some packages as delivered to see how that affects the final report.