Pratical Exercices N° 5 - Debriefing

Container Manifest

A container ship is a ship that, as the name suggests, transports containers from one port to another. Each container ship has a manifest, which is a paper document containing a list of all containers it carries. The purpose of this TP is to model this paper document.

1. **Define the `Container` type in `fr.uge.manifest`.**

```
package fr.uge.manifest;

import java.util.Objects;

public record Container(String destination, int weight){
    public Container {
        Objects.requireNonNull(destination, "destination must be not null");
        if (weight <= 0) {
            throw new IllegalArgumentException("weight must be positive");
        }
    }
}
```

   To define the `Container`, a record is the best way because we don't want to modify a container data.

2. **Define the `Manifest` type in `fr.uge.manifest` which contains a list of containers.**

```
package fr.uge.manifest;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Objects;

public class Manifest {
    private final ArrayList<Item> containers;

    public Manifest() {
        containers = new ArrayList<>();
    }

    public void add(Item container) {
        Objects.requireNonNull(container, "container must not be null");
        containers.add(container);
    }

}
```

To define the Manifest, we have to use a class instead of a record because we don't want anyone to modify list of the containers. With a record, we can acces to the container list.

3. **Print the content of containers**

```
public class Manifest {
    ...
    public String toString() {
        var output = new StringBuilder();
        var refactor = "";

        var i = 1;
        for (var item : containers) {
            output.append(refactor).append(i).append(".").append(item);
            refactor = "\n";
            i++;
        }

        return output.toString();
    }
}
```

To perform the print of containers in the manifest, we have to redifine the `toString()` in records `Container`.

```
public record Container(String destination, int weight){
    ...
    @Override
    public String toString() {
        return destination + " " + weight + "kg";
    }
}
```

4. **Add `Passenger` type to the containers**

To perform this change, we have to create an interface and implement this interface.

```
// Item.java
public sealed interface Item permits Container, Passenger {

    int weight();

    String destination();

}
```

```java
// Passsenger.java
package fr.uge.manifest;
import java.util.Objects;
public record Passenger(String destination) implements Item {
    public Passenger {
        Objects.requireNonNull(destination, "destination must be not null");
    }

    @Override
    public String toString() {
        return destination + " (passenger)";
    }

    @Override
    public int weight() {
        return 0;
    }
}
```

So have to implements the `Item` interface

```java
public record Container(String destination, int weight) implements Item{
    ...
}
```

5. **Add `price()` method to `Manifest`**

   We had to add `onBoardPrice()` to `Item` interface and implements him in the `Passenger` and `Container` class.

```java
public sealed interface Item permits Container, Passenger {
    ...
    int onboardPrice();
}
```

```java
public class Manifest {
    ...
    public int price() {
        var totalPrice = 0;
        for (var container : containers) {
            totalPrice += container.onboardPrice();
        }

        return totalPrice;
    }
}
```

6. Add `removeAllContainersFrom(destination)` method to the class.

```java
public class Manifest {
    ...
    public void removeAllContainersFrom(String destination) {
        Objects.requireNonNull(destination, "destination must not be null");

        var iterator = containers.iterator();
        while (iterator.hasNext()) {
            var item = iterator.next();
            if (item.destination().equals(destination) && item.isContainer()) {
                iterator.remove();
            }
        }
    }
}
```

7. Because the code is no longer maintainable if we add a new type to our interface.

8. **Add `weightPerDestination` method to the `Manifest`.**

```
public class Manifest {
    ...
    public HashMap<String, Integer> weightPerDestination() {
        var weightSummary = new HashMap<String, Integer>();
        for (var item : containers) {
            if (item.isContainer()) {
                var previousValue = weightSummary.getOrDefault(item.destination(),
0);
                weightSummary.put(item.destination(), previousValue +
item.weight());
            }
        }

        return weightSummary;
    }
}
```

Code Source

`Container.java`

```java
package fr.uge.manifest;
import java.util.Objects;

public record Container(String destination, int weight) implements Item {
    public Container {
        Objects.requireNonNull(destination, "destination must be not null");
        if (weight <= 0) {
            throw new IllegalArgumentException("weight must be positive");
        }
    }

    @Override
    public String toString() {
        return destination + " " + weight + "kg";
    }

    @Override
    public int onboardPrice() {
        return weight * 2;
    }

    @Override
    public boolean isContainer() {
        return true;
    }
}
```

`Item.java`

```java
package fr.uge.manifest;
public sealed interface Item permits Container, Passenger {

    int weight();

    String destination();

    int onboardPrice();

    default boolean isContainer() {
        return false;
    }

}
```

`Passenger.java`

```java
package fr.uge.manifest;
import java.util.Objects;
public record Passenger(String destination) implements Item {
    public Passenger {
        Objects.requireNonNull(destination, "destination must be not null");
    }

    @Override
    public String toString() {
        return destination + " (passenger)";
    }

    @Override
    public int weight() {
        return 0;
    }

    @Override
    public int onboardPrice() {
        return 10;
    }
}
```

`Manifest.java`

```java
package fr.uge.manifest;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Objects;

public class Manifest {
    private final ArrayList<Item> containers;

    public Manifest() {
        containers = new ArrayList<>();
    }

    public void add(Item container) {
        Objects.requireNonNull(container, "container must not be null");
        containers.add(container);
    }

    public int weight() {
        var totalWeight = 0;
        for (var container : containers) {
            totalWeight += container.weight();
        }

        return totalWeight;
    }

    public int price() {
        var totalPrice = 0;
        for (var container : containers) {
            totalPrice += container.onboardPrice();
        }

        return totalPrice;
    }

    public void removeAllContainersFrom(String destination) {
        Objects.requireNonNull(destination, "destination must not be null");

        var iterator = containers.iterator();
        while (iterator.hasNext()) {
            var item = iterator.next();
            if (item.destination().equals(destination) && item.isContainer()) {
                iterator.remove();
            }
        }
    }

    public HashMap<String, Integer> weightPerDestination() {
        var weightSummary = new HashMap<String, Integer>();
        for (var item : containers) {
            if (item.isContainer()) {
                var previousValue = weightSummary.getOrDefault(item.destination(), 0);
                weightSummary.put(item.destination(), previousValue + item.weight());
            }
        }

        return weightSummary;
    }

    @Override
    public String toString() {
        var output = new StringBuilder();
        var refactor = "";

        var i = 1;
        for (var item : containers) {
            output.append(refactor).append(i).append(".").append(item);
            refactor = "\n";
            i++;
        }

        return output.toString();
    }

}
```