Object-Oriented Language and Theory

Lab 10: Exception Handling

* Objectives:

In this lab, you will practice with:

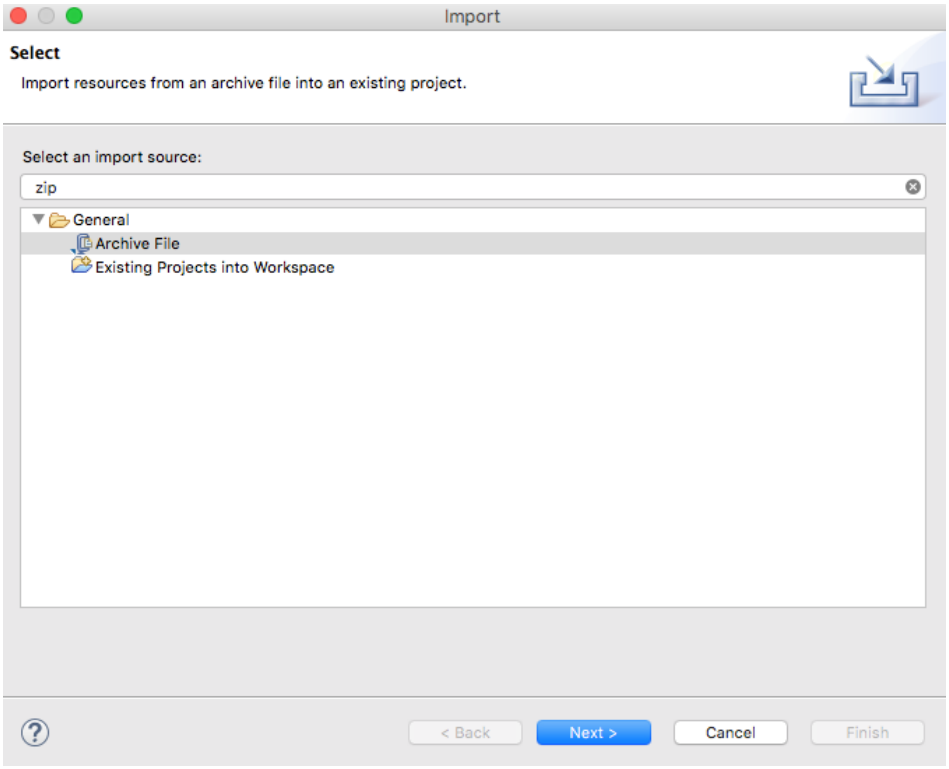
- Create various Exception types
- Raise exceptions
- Catch and report exceptions

In this lab, you will create a subclass of **`Exception`** called **`PlayerException`**. This exception is raised when one of the **`Media`** subclasses' **`play`**() method encounters a **`length`** of 0. The **`play`**() method will be altered to use **`try-catch`** syntax to catch the error.

0. Open the workspace and the AIMS project

- Open Eclipse

- Open File -> Import. Type zip to find Archive File if you have exported as a zip file before. You may choose Existing Projects into Workspace if you want to open an existing project in your computer. Ignore this step if the AimsProject is already opened in the workspace.



- Click Next and Browse to a zip file or a project to open
- Try to apply Release Flow by yourself.

1. Check all the previous source codes to catch/handle/delegate runtime exceptions

Review all methods, classes in AimsProject, catch/handle or delegate all exceptions if necessary. The exception delegation mechanism is especially helpful for constructors so that no object is created if there is any violation of the requirement/constraints.

Hint: In Aims Project, we can apply exception handling to *validate data constraints* such as non-negative price, to *validate policies* like the restriction of the number of orders, and to handle *unexpected interactions*, e.g., user try to remove an author while the author is not listed.

For example, the following piece of code illustrates how to control the number of orders with exception.

```
public Order() throws LimitExceededException {
	if (Order.nbOrders < MAX_LIMITTED_ORDERS) {
		// TODO Set initial values for object attributes
	} else {
		throw new LimitExceededException("ERROR: The number of orders has
reached its limit!");
	}
}
```

2. Create a class which inherits from Exception

The **`PlayerException`** class represents an exception that will be thrown when an exceptional condition occurs during the playing of a media in your **`AimsProject`**.

2.1. Create new class named `PlayerException`

- Enter the following specifications in the New Java Class dialog:

- Name: **`PlayerException`**
- Package: **`hust.soict.ictglobal.aims`**
- Access Modifier: **`public`**
- Superclass: **`java.lang.Exception`**
- Constructor from Superclass: checked
- **`public static void main(String [] args)`:** do not check
- All other boxes: do not check

- Finish

2.2. Raise the `PlayerException` in the `play()` method

- Update **`play()`** method in **`DigitalVideoDisc`** and **`Track`**

- For each of **`DigitalVideoDisc`** and **`Track`**, update the **`play()`** method to first check the object's length using **`getLength()`** method. If the length of the **`Media`** is less than or equal to zero, the **`Media`** object cannot be played.
- At this point, you should output an error message using **`System.err.println()`** method and the **`PlayerException`** should be raised.

- The example of codes and results for the **play**() of **DigitalVideoDisc** in **JavaFX** are illustrated in the following figures.

```
public void play() throws PlayerException {
	if (this.getLength() > 0) {
		// TODO Play DVD as you have implemented
	} else {
		throw new PlayerException("ERROR: DVD length is non-positive!");
	}
}
```

- Save your changes and make the same with the play() method of Track.

2.3. Update play() in the Playable interface

- Change the method signature for the **Playable** interface's **play()** method to include the throws **PlayerException** keywords.

2.4. Update play() in CompactDisc

- The **play**() method in the **CompactDisc** is more interesting because not only it is possible for the **CompactDisc** to have an invalid **length** of 0 or less, but it is also possible that as it iterates through the tracks to play each one, there may have a track of length 0 or less
- First update the **play**() method in **CompactDisc** class to check the length using **getLength**() method as you did with **DigitalVideoDisc**
- Raise the **PlayerException**. Be sure to change the method signature to include **throws PlayerException** keywords.
- Update the **play()** method to catch a **PlayerException** raised by each **Track** using block **try-catch.**

The code example is shown as follows.

```
public void play() throws PlayerException{
	if(this.getLength() > 0) {
		// TODO Play all tracks in the CD as you have implemented
		java.util.Iterator iter = tracks.iterator();
		Track nextTrack;
		while(iter.hasNext()) {
			nextTrack = (Track) iter.next();
			try {
				nextTrack.play();
			}catch(PlayerException e) {
				throw e;
			}
		}
	}else {
		throw new PlayerException("ERROR: CD length is non-positive!");
	}
}
```

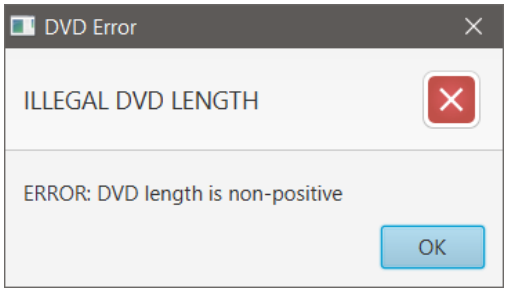
- You should modify the above source code so that if any track in a **CD** can't play, it throws a **PlayerException** exception.

3. Update the Aims class

- The **Aims** class must be updated to handle any exceptions generated when the **play()** methods are called. What happens when you don't update for them to catch?
- Try to use **try-catch** block when you call the **play()** method of **Media's** objects.

With all these steps, you have practiced with User-defined Exception (**PlayerException**), **try-catch** block and also **throw**. The **try-catch** block is used in the main method of class **Aims.java** and in the **play()** method of the **CompactDisc.java**. Print all information of the exception object, e.g. **getMessage()**, **toString()**, **printStackTrace()**, display a dialog box to the user with the content of the exception.

The example of codes and results for the **play**() of **DigitalVideoDisc** in **JavaFX** are illustrated in the following figure.



4. Modify the equals() method and compareTo() method of Comparable for Media class:

- Two medias are equals if they have the same **title** and **cost**
- Please remember to check for **NullPointerException** and **ClassCastException** if applicable.

You may use **instanceof** operator to check if an object is an instance of a **ClassType**.

5. Reading Document

Please read the following links for better understanding.

- Exception-handling basics: https://developer.ibm.com/tutorials/j-perry-exceptions/
- Basic guidelines: Although the examples are in C++, the ideas are important. https://docs.microsoft.com/en-us/cpp/cpp/errors-and-exception-handling-modern-cpp?view=vs-2019#basic-guidelines

6. Exercises

- Update policy to get a lucky item:
    - Now, there is a fixed probability to get a lucky item or nothing

    - To have a chance to get a lucky item, the total price and the number of items must be higher than pre-defined thresholds. For instance, an order of $322 with 7 items will has a chance to get a lucky item, but an order of $420 with 2 items or an order of $9 with 10 items will be not allowed to get a lucky item.
    - The higher the total price of an order is, the higher the value of lucky item can be; however, it must be always less than a pre-defined threshold. To illustrate, an order of $177 can have a lucky item up to $50, an order of $352 can have a lucky item up to $100, and an order of $1000 can have a lucky item up to $100.
- Make an exception hierarchical tree for all self-defined exceptions in Aims Project. Use class diagram in Astah to draw this tree, export it as a png file, and save them in design directory.