Project: The JukeBox

Objectives

- Summative evaluation of all concepts, knowledge & skills learnt in the Java program
- Apply clean coding and standards & guidelines to Demonstrate the software engineering competency
- Use the professional skills learnt during the program to prepare and present your project to stakeholders

The Jukebox

Jukeboxes were invented even before digital music came into existence – and certainly much before streaming music became available.

Music jukeboxes were commonly used in restaurants and bars to enable customers to choose what music they wanted to listen to.

Electronic and digital jukeboxes became prevalent from the 70’s and 80’s. Modern jukeboxes are fully computerized and play stream music from the internet directly.

The JukeBox Challenge

- Let us design a modern jukebox that allows users to listen to music and podcasts that are live streamed. There are a lot of tracks and episodes on the jukebox. So, whether you're behind the wheel, working out, partying or relaxing, the right music or podcast is always at your fingertips. You can also browse through the artists, and celebrities, or create a playlist and just sit back.

- In this challenge you will design a modern jukebox that contains songs by various artists and podcasts by celebrities.

- Project scope is divided into 4 tasks to be done over 24 hours

Task 1

- The jukebox contains a catalog of songs by various artists. Add different genres of songs into the catalog.
- The catalog can be categorized based on the artist, genre or name of the album.
- The home page of the jukebox displays all the songs available in the catalog.
- Ability to search for music based on any of the categories mentioned in point 2. Once a category is specified, an alphabetical search can be offered to the user to locate the music they are looking for.

Task 2

- The modern jukebox allows users to create a catalog of podcasts and to live stream music of various artistes.
- The homepage of the jukebox displays podcasts by celebrities.
- Allows a user to search for a podcast by celebrity or date on which the podcast was published.

Task 3

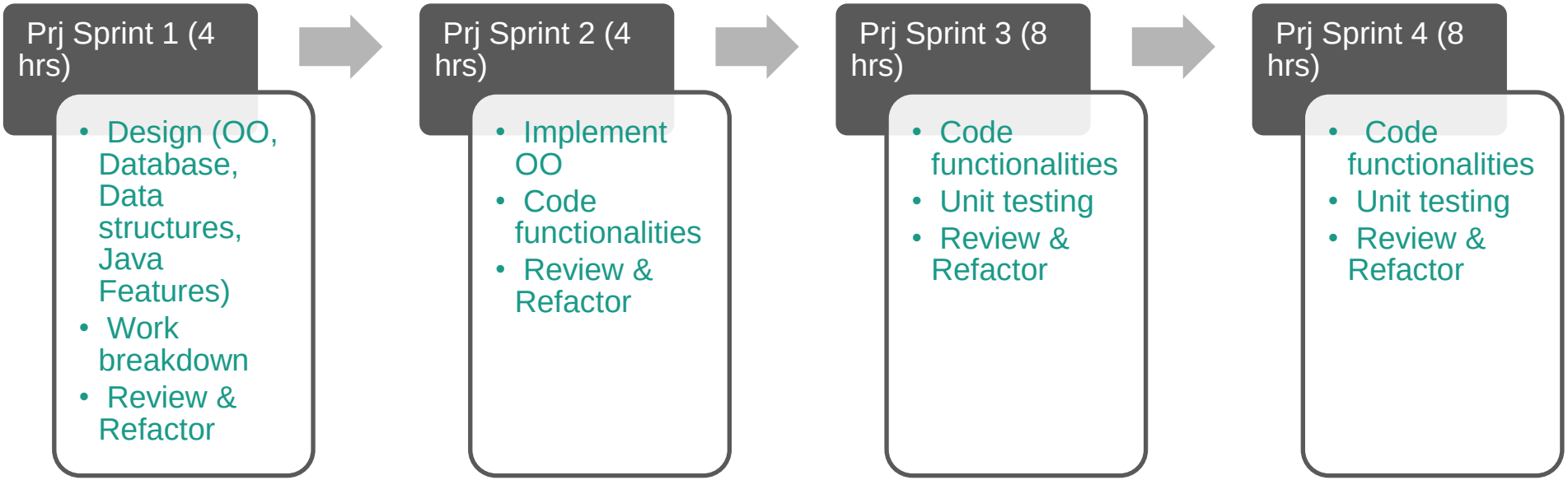
- The jukebox allows a user to either create a playlist of songs or podcasts or a combination of the two.
- Allows the user to add a song or a particular album or a podcast to the created playlist.
- The user can create multiple playlists and can view the contents of the playlists.

Task 4

- The design of the modern jukebox has been completed, now a user can play songs from the created playlists.
- Display all the songs in the playlist along with the name of the song, album name if any and the duration of the song in a tabular format.
- The user must be able to play any song or podcast from the playlist.
- The current song that is playing with the time remaining in mm:ss format and the list of songs that have been queued up for playing subsequently.
- The user must be able to perform reverse, forward, pause, resume and shuffle or loop play of the songs in the playlist.
- Hint : Use the AudioInputStream class to play the songs
- Reference - https://docs.oracle.com/en/java/javase/11/docs/api/java.desktop/javax/sound/sampled/AudioInputStream.html

Project Development - Approach

Project Sprint – Suggested Approach



Sprint 1: Design Discussion

- Read the requirements from the problem statement and create project design
- Divide them into various tiers
- Identify all the classes their attributes and behaviours, relationship among classes, reusability of methods,
- Demonstrate the usage of data within the project, If RDBMS is used then represent by using E-R diagram
- Demonstrate the flow of project
- How will the users interact with working on the Project?

Sprint 2: Implement OO Code functionalities Review & Refactor

- Model the classes, attributes and behaviour identified in the design phase
- Provide structure to the classes
- Identify the OOPs concepts that can be used
- Implement the OOPs concepts in the class structure created
- Add all the required attributes inside each classes
- Use property methods wherever required
- Add the behaviours

Sprint 3: Code functionalities Unit testing Review & Refactor

- Implement the behaviours
- Implement appropriate access specification
- Write the business logic in the methods
- Ensure appropriate usage of
    - Conditions and Looping
    - Arrays, Strings and Regular Exp
    - Predefined and Custom Exceptions
    - Data Structure
    - Collections and Java8 features

Sprint 4: Code functionalities Unit testing Review & Refactor

- Implement File handling if required
- Reading the data from file
- Writing to the file
- Implement JDBC if required
- CRUD operations
- Implement Unit testing by using JUNIT

Final Refactoring and Closure

- Fixing errors
- Fixing bugs
- Check Lint errors
- Ensure that project is running

Let’s Recall

Recall

- OOP concepts are the building blocks of the Java Language.
- Applying OOAD to implement a design of the project is crucial
- Designing the application before implementation is important and reduces the effort of refactoring.
- The principles of OOP
    - Encapsulation
    - Abstraction
    - Inheritance
    - Polymorphism

Recall

- The Java Naming conventions
- Notations in identifiers
- Conditional and looping constructs
- Appropriate usage of Data structures

Recall

- Exception Handling – a mechanism to handle the abnormal termination of a program.
- Collections – An API to structure data
- Java 8 Features – The new features provided by Java
  - Lambdas
  - Functional Interfaces
  - Streams
  - Optional

Recall

- TDD – Test Driven Development
  - Testing applications is as important as developing them.
  - Testing helps a developer to verify code by providing multiple input parameters
- Java programs can be tested using JUNIT
- The JUnit Platform serves as a foundation for launching testing frameworks on the JVM.
- It also defines the Test Engine API for developing a testing framework that runs on the platform.
- PMD checks in a java program

Bad Coding Practices

Bad Practices

- PMDs are a set of rules that do a quality check on the source code written.
- It finds common programming flaws like unused variables, empty catch blocks, unnecessary object creation, unused imports, unnecessary print statements.

```
public static void main(String[] args) {
 String name = "Gary";
 int age = 20;
 System.out.println(name);
```

Age variable is never used

```
int arr[] = {10,15,25,5,50,45,35};
for(int i:arr)
    System.out.println(i);
```

No {} brackets for the block

```
try{
    //statements
}
catch(Exception e){

}
```

Empty catch block

Bad Practices

- Avoid hard coded values

```
int arr[] = {10,15,25,5,50,45,35};

for(int i = 0; i < 7;i++ )
    System.out.println(i);
```

- In the above code the array length is hard coded in the for loop.

Bad Practices

Unnecessary or excessive usage of looping and decision making constructs.

```
}
else if(sourceCurrency==1 && targetCurrency==3)
{
    convertedAmount = (amountToBeConverted*JPY/GBP);
}
else if(sourceCurrency==1 && targetCurrency==4)
{
    convertedAmount=amountToBeConverted*JPY/USD;
}
else if(sourceCurrency==1 && targetCurrency==5)
{
   convertedAmount=(amountToBeConverted*JPY/AED);
}
else if(sourceCurrency==2 && targetCurrency==1)
{
      convertedAmount=(amountToBeConverted*EUR/JPY);
}
else if(sourceCurrency==2 && targetCurrency==3)
{
    convertedAmount=(amountToBeConverted*EUR/GBP);
```

Bad Practices

- Avoid creating collection objects inside loops

```
for(int i = 0; i < 7;i++)
{
    List<Employee> empList = new ArrayList<>();
}
```

Bad Practices

- The test case passes but is not testing the actual feature.
- Testing multiple things in assertions in a single test case.
- The test case is handling an exception.
- The setUp and tearDown methods not written.
- Individual test cases must not do set up and tear down operations like initialization of values, creation of objects etc.

Best Coding Practices

Best Practices

- Ensure the program has all the PMD checks in place.
- Ensure modularity of the application.
- Write comments to explain the functionality of the method and class.
- Ensure meaningful class and method names are used
- The IS-A and HAS-A relationship must be correctly identified between classes.
- Use functions like length or size to loop through arrays or collections.
- Ensure the code written is maintainable, use Java 8 features to reduce the lines of code.

Best Practices

- Use a TDD approach when developing the application.
- Ensure that an equal measure of positive and negative test scenarios are written.
- Use assertion statements appropriately.
- Ensure the test case actually tests the functionality.
- Ensure that the test cases have setUp and tearDown methods for initialization and object creation purposes.

Common Mistakes during Development

Common Mistakes

- Common mistakes while implementing functional logic:
  - Poor design during the OOAD phase
  - Creation of unnecessary classes
  - Classes and methods created do not have a meaningful name.
  - Writing all classes in a single package
  - Too many custom exceptions created
  - Too many looping and conditional constructs
  - Bad usage of lambda and functional interfaces
- Common mistakes while implementing test cases:
  - Test cases do not convey the testing intent
  - Too many assertion statements in a single test case

Think and Tell

- In real work, how does a project gets evaluated? And therefore the developer's performance.
- Is it enough if the project works fine (functionality wise)?
- Is there a single score like any exam?
- What are parameters customer and delivery team look for?

Evaluation Parameters: Rubrics

Parameter	Description
Project Design	Design (OO decomposition, Database design, Data Structures) and break down of the work plan
Apply Core Java concepts, knowledge and skills	Usages of OOPS Features
	Usage of Data Structure Algorithms and Java Collections
	Java 8 features(lambdas, Streams classes)
	Manage Data By Using an appropriate source
Software Engineering Practices	Junit testing
	Code Quality (Efficient uses of loops, decision making, appropriate use of collections classes)
	Styles and Guidelines (Naming conventions for classes, variables and methods
Project Completion	Project Submission (including review and refactor)
	Functional Completeness
Show & Tell	Project Presentation