Tabish Parkar

Formative Assessment 1:

Systems Development 2A

1.1. Object-oriented inheritance allows us to create a series of classes that shows various types of books. The base class in this case would be “Book” which would encapsulate the shared characteristics and methods that is common to all books. The “FictionBook” subclass will inherit these characteristics while also adding its specific attributes. Inheritance also improves code management by having a clear structure that separates general and specific functionalities. The result of this is that it promotes code reusability which overall maintains consistency and minimizing redundancy throughout the codebase.

1.2. To derive a subclass named “FictionBook” from the class “Book” we need to verify that the “Book” class is made correctly to actually allow inheritance.

```java
public class Book {
    private String title;
    private String author;
    private String isbnNumber;
    private boolean isAvailable;

    public Book(String title, String author, String isbnNumber) {
        this.title = title;
        this.author = author;
        this.isbnNumber = isbnNumber;
        this.isAvailable = true;
    }

    public String getTitle() {
        return title;
    }

    public String getAuthor() {
        return author;
    }

    public String getIsbnNumber() {
        return isbn;
    }

    public boolean isAvailable() {
        return isAvailable;
    }

    public void markAsBorrowed() {
        this.isAvailable = false;
    }

    public void markAsReturned() {
        this.isAvailable = true;
    }
}

public class FictionBook extends Book {
```

```java
public class FictionBook extends Book {
    private String genre;

    public FictionBook(String title, String author, String isbnNumber, String genre) {
        super(title, author, isbnNumber);
        this.genre = genre;
    }

    public String getGenre() {
        return genre;
    }

    public void printQuickDescription() {
        System.out.println("Title: " + getTitle() + ", Author: " + getAuthor() + ", Genre: " + genre);
    }
}

public class LibrarySystem {
    public static void main(String[] args) {
        FictionBook fictionBook = new FictionBook("Harry Potter And The Goblet Of Fire", "J.K. Rowling", "978-0547928227", "Fantasy");
        fictionBook.printQuickDescription();
    }
}
```

“FictionBook” will then carry over the qualities of “Book” class and will also add other qualities to achieve the needed goals of the library.

2.1. Exceptions in Java are events obstacles the normal process of a program. They are connected to file-handling processes because file input or output which could follow to multiple constraints. Handling these issues using try-catch blocks enabling engineers to mitigate errors and better the overall user experience by giving information driven error messages instead of the whole app crashing.

2.2.

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class FileReadingExample {
    public void readFile(String fileName) {
        BufferedReader reader = null;

        try {
            reader = new BufferedReader(new FileReader(fileName));
            String line;
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            System.err.println("Error reading file: " + e.getMessage());
        } finally {
            try {
                if (reader != null) {
                    reader.close();
                }
            } catch (IOException e) {
                System.err.println("Error closing file: " + e.getMessage());
            }
        }
    }
}
```

2.3.

```java
import java.nio.file.*;

public class FileOperations {
    public void copyFile(String source, String destination) {
        try {
            Path sourcePath = Paths.get(source);
            Path destinationPath = Paths.get(destination);
            Files.copy(sourcePath, destinationPath, StandardCopyOption.REPLACE_EXISTING);
            System.out.println("File copied successfully.");
        } catch (IOException e) {
            System.err.println("Error during copy: " + e.getMessage());
        }
    }
}
```

2.4. File organization refers to the way data is built in files so that it can be efficiently opened and changed. In the context of file handling, streams are sequences of data being written or read from files. Java has different types of streams to manage the reading and writing methods.

Buffers don’t always keep data during I/O operations to better efficiency along with reducing the amount of reading or writing calls made to the system. Having buffered streams in Java improves performance, mainly when dealing with large file sizes or constant input or output operations.

2.5.

```java
import java.io.RandomAccessFile;
import java.io.IOException;

public class RandomAccessFileExample {

    public void writeRecord(String filePath, String record) {
        try (RandomAccessFile raf = new RandomAccessFile(filePath, "rw")) {
            raf.seek(raf.length());
            raf.writeBytes(record + "\n");
            System.out.println("Record written successfully.");
        } catch (IOException e) {
            System.err.println("Error writing record: " + e.getMessage());
        }
    }
}
```

Using the RandomAccessFile class in the example shown above, enables reading from and writing to files at any location.

RandomAccessFile also enables for a person to find to any piece of the file to read or write data, which in turn makes it a perfect option for applications where the user is needed to use data records.