CSC207 – Quiz 2 SOLUTIONS

1. Here is part of the UML diagram for the attached code. Complete the diagram.

**Note:** For this quiz only, you can use bold text instead of italic text if you have difficulty writing text that is clearly italic.

![UML Diagram](image)

(insert dotted upwards arrow here) (insert two solid upwards arrows here)
2.(a) Write a main method that creates a BookShelf that stores objects of type Book. It should contain a TextBook called “The UNIX Time-Sharing System” by “Ritchie” and “Thompson”.

The main method should also create a BookShelf of Strings, where each String is the title of one of the following books: “Ender’s Game”, “Dune”, and “Snow Crash”. Your code should add each String to the BookShelf in the order they appear on the list. However, the BookShelf’s contents should store them in alphabetical order.

One possible solution:

```java
public static void main(String[] args) {
    ArrayList<String> unixAuthors = new ArrayList<String>();
    unixAuthors.add("Ritchie");
    unixAuthors.add("Thompson");
    TextBook unixBook = new Textbook("The UNIX Time-Sharing System", unixAuthors);
    BookShelf<Book> demoBookShelf = new BookShelf<Book>(unixBook);

    BookShelf<String> demoStringBookShelf = new BookShelf<String>("Ender's Game");
    demoStringShelf.addLeft("Dune");
    demoStringShelf.addRight("Snow Crash");
}
```

2.(b) Consider the following code contained in a main method that is not in any of the given classes:

1. ComicBook cm1 = new ComicBook("SuperHero", "Comic Author");
2. Shelf<Object> shelf0 = new BookShelf<Object>(cm1);
3. System.out.println((shelf0.getLeft()).getCreators());

This code will prevent the program from running. Explain the problem and suggest two different ways to fix it. Refer only to these three lines of code and the supplementary code at the end of this booklet.

Solution:

On line #3, the method `getLeft()` will return an value of type `Object` which does not have a `getCreators()` method. This can be fixed by either:

(i) changing line #2 to say: Shelf<ComicBook> shelf0 = new BookShelf<ComicBook>(cm1); or Self<Book> shelf() = new BookShelf<Book>(cm1);

(ii) casting on line #3: System.out.println(((ComicBook) shelf0.getLeft()).getCreators());
3. (a) Write the code for a class called `MuseumCase` that implements interface `Shelf` and a class called `Statue` that can be stored in a `MuseumCase`. A `MuseumCase` can store a maximum of 20 instances of `Statue`. It should not be a generic class; instead, it should always only contain `Statue` objects. Class `Statue` contains a `String` called `description` and a `int` called `age`, a constructor, methods for getting and setting `description`, getting (but not setting) `age`, and a `toString()` method that returns the description and age together in a `String` such as "Osiris 5000". You do not need to write a `main` method for either class.

```java
public class Statue {
    private String description;
    private int age;

    public Statue(int age) {
        this.age = age;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public int getAge() {
        return age;
    }

    @Override
    public String toString() {
        return description + " " + age;
    }
}
```

```java
public class MuseumCase implements Shelf<Statue> {
    private LinkedList<Statue> contents = new LinkedList<Statue>();

    public MuseumCase() {
    }

    public MuseumCase(Statue statue) {
        contents.add(0, statue);
    }

    public void addLeft(Statue statue) {
        if (contents.size < 20) {
            contents.add(0, statue);
        } else {
            System.out.println("Museum Case is full.");
        }
    }
}
```
public void addRight(Statue statue) {
    if (contents.size < 20) {
        contents.add(statue);
    } else {
        System.out.println("Museum Case is full.");
    }
}

public Statue getLeft() {
    return contents.get(0);
}

public Statue getRight() {
    return contents.get(contents.size() - 1);
}

3.(b) What are two big differences between class MuseumCase and BookShelf?

(i) MuseumCase has a fixed capacity while Bookshelf does not.

(ii) MuseumCase only stores variables of type Statue, while BookShelf is generic and can store any type.

3.(c) Why is Book an abstract class and not an interface?

Class Book contains methods that are not static, but have been implemented. This is allowed for an abstract class, but not an interface.
Supplementary Code – Quiz 2

The questions in the comments of the following code are **not** for marks.

```java
public interface Shelf<T> {

    /** Adds shelfItem to the beginning of the shelf. */
    public void addLeft(T shelfItem);

    /** Adds shelfItem to the end of the shelf. */
    public void addRight(T shelfItem);

    /** Returns the left-most item on the shelf. */
    public T getLeft();

    /** Returns the right-most item on the shelf. */
    public T getRight();
}

import java.util.LinkedList; // LinkedList replaced ArrayList

public class BookShelf<T> implements Shelf<T> {

    private LinkedList<T> contents = new LinkedList<>();
    // contents is now a local variable in a method.

    public BookShelf() {
    }

    public BookShelf(T item) {
        contents.add(0, item);
    }

    public void addLeft(T item) {
        contents.add(0, item);
    }

    public void addRight(T item) {
        contents.add(item);
    }

    public T getLeft() {
        return contents.get(0);
    }

    public T getRight() {
        int numBooks = contents.size();
        return contents.get(numBooks - 1);
    }
}
```
public abstract class Book {

    private String title;
    protected static String itemGroup = "Book"; // itemGroup used to be public.

    public Book() {
    }

    public Book(String title) {
        this.title = title;
    }

    public String getTitle() {
        return title;
    }

    public abstract String getCreators();

    protected static String getItemGroup() {
        return itemGroup;
    }
}

import java.util.ArrayList;

public class TextBook extends Book {

    private ArrayList<String> authors = new ArrayList<>();
    protected int edition = 1;

    public TextBook() {
    }

    public TextBook(String title, ArrayList<String> authors) {
        super(title);
        this.authors = authors; // This points to the parameter ArrayList instead of copying it.
        // What happens if the parameter list is changed elsewhere?
    }

    public void setEdition(int ed) {
        if(ed > 0) {
            edition = ed;
        }
    }

    public String getCreators() {
        return authors.toString();
    }
}
public class ComicBook extends Book {

    private String author;
    private String artist;

    public ComicBook() {
    }

    public ComicBook(String title, String author) {
        this(title, author, null);
    }

    public ComicBook(String title, String author, String artist) {
        super(title);
        this.author = author;
        this.artist = artist;
    }

    public String getCreators() {
        return "Author: " + author + ", Artist: " + artist;
    }
}
