Teesside University | International Study Centre | StudyGroup

TUISC - Assessment Brief – Java Programming Summative 3 Re-sit

Title of the assignment:	Fun Park Prototype		
Programme:	International Year 1 - Computing		
Assessment type (ICA/ECA):	ECA	**Academic Year:**	2021-2022
Module Title:	Java Programming	**Module Code:**	SGL0008-N
Assignment Format & Maximum Word Count:	Prototype application, documentation and report (report 500 max word count)	**Weighting:**	55%
Submission Date/Time and Location/Method:	04/08/22 at 5pm	**Tutor/s Marking:**	Julie King
Feedback date:	18/08/22	**Resubmission Date:**	

Detailed Brief for Individual/Group Assessment

Purpose of this document:

This document describes the resit assessment for the module. It is an end of course assignment ECA) with a single submission. This is an individual ECA.

Context of the assignment:

Background – Fun Park

A local fun park offers three activities – climbing, trampolining and water slides. When customers enter the park, they can purchase access to as many of the available activities as they would like to take part in that day. The prices for each activity are listed in the table below.

Activity	Price
Climbing Wall	£10
Trampolining	£12
Water Slides	£15

If customers pay for one activity, they will pay full price. For each additional activity purchased, the customer will receive a discount: For two activities they will receive a 10% discount on their total price. For three activities, they will receive a 20% discount on the total price.

As well as entering the inputs identified above, the customer's name should also be input (e.g. "J Smith" up to a maximum of 25 characters). Once the validation of inputs and cost calculation has been completed, an e-ticket should be output as below:

Customer: J Smith

Date: 19 Jul 2022 Discount: 10%

Activity 1: Climbing

Activity 2: Water Slides

Activity 3:

Price including discount: £22.50

Figure 1: E-ticket Output

If the user purchases access to just one activity, the ticket should output without the discount section.

Part 1 – Fun Park Java Prototype (60%)

Instructions

A prototype console application (no GUIs or GUI dialogs) is required as proof concept. Your prototype should display a menu similar to the following:

```
1.    Enter new E-Ticket
2.    Display Summary of Purchases
3.    Display Summary of Purchases for Selected Month
4.    Find and display customer
0.    Exit
```

Once an option is selected and processed, the menu should be redisplayed until the exit option is selected. Each operation is described in the next set of sub-headings.

1. Enter new E-Ticket

Ask the user to input the: customer's name and select the activities they are purchasing). Note: All inputs should be validated.

- Calculate the price (including any discounts).
- Display the e-ticket (it must use the format in Figure 1.).
- Save (append) the e-ticket data to the text-based summary file (tickets2022.txt). Each line in the file must hold the details for a single e-ticket with the following information separated by commas:
    - Date (today's date – see Technical Expectations section for format).
    - Activity 1 ('C' for Climbing Wall, 'T' for Trampolining, 'W' for Water Slides)
    - Activity 2 ('C' for Climbing Wall, 'T' for Trampolining, 'W' for Water Slides or '0' if only one activity purchased)
    - Activity 3 ('C' for Climbing Wall, 'T' for Trampolining, 'W' for Water Slides or '0' if only one activity purchased)
    - Price (in pence) the raw figure *after* discounts.
    - Customer name.

2. Summary of Purchases

Prompt the user to select last year's (tickets2021.txt) or new (tickets2022.txt) ticket purchases, then show a summary including:

- Total number of tickets purchased.
- Average ticket price.
- Number of tickets sold for each activity (Climbing Wall, Trampolining, and Water Slides).
- Total number of tickets per month (assume the tickets2022.txt file only includes data for the current year). Show an entry for all twelve months, even those that have an entry of zero.

Sample Output (note figures not accurate)**:**

```
Summary of File: tickets2021.txt

Total tickets purchased: 52
Average ticket price: £24.50

Tickets purchased per activity:
Climbing Wall: 15
Trampolining: 17
Water Slides: 20

Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov  Dec
 0    0    0    0    0    0   44    8    0    0    0    0
```

3. Summary of Purchases for a given month

User is asked to select 2021 (tickets2021.txt) or 2022 (tickets2022.txt) purchases, a valid month and a summary of the following is displayed:

- Total number of purchases.
- Average ticket price
- Number of tickets sold for each activity (climbing, trampolining and water slides).

Sample Output (note figures not accurate):

```
Summary of File: tickets2021.txt

Total tickets purchased in July: 44
Average ticket price: £23.50

Tickets purchased per activity:
Climbing Wall: 13
Trampolining: 12
Water Slides: 19
```

4. Find and Display Customer

Prompt the user to select 2021 (tickets2021.txt) or 2022 (tickets2022.txt) ticket sales, then enter text to search. The selected file should be searched for any matches (full or partial) against the customer's name. For example, entering NS would match against "F Townsend".

For *each* matching entry, the relevant e-ticket (see Figure 1 Summary Output) should be displayed.

0. Exit

Display an appropriate message and exit the application.

Technical Expectations

The prototype must be developed in NetBeans.

Data files have been provided and must be named `tickets2021.txt` and `tickets2022.txt`. The format of the files is identical.

Validation on all user inputs is a must. As well as being correct, it must also be user friendly by concisely informing the user of the error and allowing a re-entry opportunity.

Your code must be well documented throughout. For top marks you must document your code using Javadoc. For more information see this Oracle tutorial: https://www.oracle.com/uk/technical-resources/articles/java/javadoc-tool.html

To make your code easily readable by your tutor, you must adhere to the Google Java style guide - https://google.github.io/styleguide/javaguide.html. Deviation from this guide will result in lower marks.

The implementation must include a `Ticket` class that represents a single e-ticket for up to three activities at a time. You must also have a driver class named `FunPark`. You may create further classes / subclasses as necessary for your implementation. You must demonstrate knowledge of basic object-oriented concepts including classes, objects, methods, and encapsulation.

The following method will get today's date:

```
public static String getDate()
    {
        Calendar cal = Calendar.getInstance();
        SimpleDateFormat sdf = new SimpleDateFormat("dd-MMM-yyyy");
        return sdf.format(cal.getTime());
    }
```

You will need to import `java.util.Calendar` and `java.text.SimpleDateFormat`

Internally use integers for monetary values (hold them as pence). Display as pounds and pence by dividing the value by 100 and using the `printf` method to display. For example:

```
System.out.printf("£%.2f ", amount / 100);
```

Guidance

Design your solution using pseudo code or flow charts. Resist the urge to start hacking away immediately.

Save frequently and backup your work every day. Work methodically and proceed using small steps.

Work individually. Your code will be checked for plagiarism.

You are **strongly advised** to **retain all versions** of your **code** and your **development notes**, so that you can show the development of your work. This may be required as evidence in the event of any query about the authorship of your work

Part 2 - Testing Documentation (20%)

Instructions

Produce a black-box test plan containing the test cases applied to your solution to Part 1. The expected format of your test cases is shown in the table below (colour scheme is not necessary but headings are):

Id	Description	Steps	Expected	Actual	Result	Comment

Guidance

You should begin this documentation the same time as you start implementation of Part 1. It is much simpler to test as you develop (also known as *integration* testing) as opposed to 'faking' it before submission.

Remember to test all cases. It is very easy to test for success, but you must also test for failure, and your boundaries (your solution is likely to make use of multiple looping constructs).

You may want to consider using alternative tools such as Microsoft Excel to make managing and organising your test plan easier as the number of test cases grow.

When testing your prototype, we will look for correlation between your test plan to the solution. You will be marked down for tests that you have "passed" if they should fail under assessment testing.

Part 3 – Report (10%)

Instructions

The client is considering implementing the full system and would like your advice on the following:

- What security measures/features would need to be implemented to ensure the data in the system is safeguarded?
- What legal considerations need to be respected before the full system becomes operational?
- How could development and implementation risks be minimised?

Current state of the prototype:

- Which parts have been successfully implemented and those parts which have not been fully successful?
- Provide a description of any known errors.

Your advice must be documented in a brief but professionally formatted report that is no more than 500 words to be submitted with the prototype.

Guidance

When referencing code, to make code examples more readable you can copy and paste Java code using Visual Studio Code (a free code editor from Microsoft) into Microsoft Word. This will bring along Java syntax highlighting, and the monospaced font Consolas.

Stick to the word limit and ensure you cover each point in equal detail. You will lose marks if you fail to address all points, no matter how much detail you add to another to compensate.

Teesside University International Study Centre StudyGroup

Marking Criteria

Criteria	Excellent (70-100%)	Very Good (60-69%)	Good (50-59%)	Satisfactory (40-49%)	Fail (0-39%)
Java solution structure [15]	Sensible scoping throughout, appropriate data structures used, with little wasted code that fully adheres to the Google Java style guide.	Sensible scoping throughout, appropriate data structures used, with little wasted code, minor deviations from Google Java style guide.	Appropriate Java packages imported, appropriate data structures used but approach may not be the most efficient, variables created succinctly, some repeated code blocks, with some deviations from the Google Java style guide.	Few Java packages imported, some data structures may be inappropriate, variables haphazardly created, contains blocks of repeating code, with many deviations from the Google Java style guide.	Code is not in a runnable state. Contains syntax errors, or unresolved runtime errors.
	Excellent use of object-oriented concepts including classes, objects, methods, and encapsulation.	Good use of basic object-oriented concepts including classes, objects, methods, and encapsulation.	Reasonable attempt to use object-oriented concepts including classes, objects, methods, encapsulation but questions over aspects of implementation.	Limited use of object-oriented concepts with most of the solution relying on a procedural approach.	Code does not make use of object-oriented programming.
	Code is excellently documented using Javadoc.	A reasonable attempt has been made to document the code using Javadoc.	Appropriate code comments are spread through the solution.	Code comments are sparse.	Few useful comments in code.
	The solution throws no uncaught exceptions.	Exception handling covers exceptions from imported packages, file verification, user input, and type casting.	Exception handling is present but limited to file verification and type checking.	Exception handling is very limited.	Exception handling is not present or left as empty blocks.
Menu and enter new e-ticket [20]	*All* aspects implemented as required. All appropriate OOP concepts are implemented correctly. Menu works as expected and e-ticket outputs in the format shown in the brief. E-ticket details are successfully saved to the correct file.	*Most* aspects implemented as required. All appropriate OOP concepts are implemented correctly. Most menu options work as expected and e-ticket outputs in a format very similar to that shown in the brief. E-ticket details are successfully saved to the correct file.	*Many* aspects implemented as required. Many appropriate OOP concepts are implemented correctly. Many menu options work as expected and e-ticket outputs in a similar format to that shown in the brief. E-ticket details are successfully saved to the correct file but may be in the incorrect format.	*Some* aspects implemented as required. Some appropriate OOP concepts are implemented correctly. Some menu options work as expected and e-ticket outputs with some of the same information to that shown in the brief. E-ticket details are successfully saved but may be in the incorrect format or in the incorrect file.	*Few or no* aspects implemented as required. Questions over use of OOP, or solution may be procedural. Logic may not be entirely correct, or erroneous. Menu may not work, or most aspects may be missing. E-ticket details are not saved to file.
Summary of purchases (including months) [15]	Appropriate file selected. Excellent logic/structure for file handling. All aspects implemented accurately.	Appropriate file selected. Good logic/structure for file handling. Most aspects implemented accurately.	Appropriate file selected. Reasonable logic/structure for file handling. Many aspects implemented accurately.	Appropriate file selected. Some attempt to provide logic/structure for file handling. Some aspects implemented accurately.	Unable to select the appropriate file. Little of no attempt to provide logic/structure for file handling. Very few or no aspects implemented accurately.

Find and display customers [10]	Successfully search file for matches; Correct customer summary displayed; error message provided if no matches found.	Most aspects successfully implemented; correct customer summary displayed; appropriate error message provided if no matches found.	Many aspects successfully implemented; correct customer summary displayed; appropriate error message provided if no matches found.	Some aspects implemented successfully; partial customer summary displayed; no or limited message provided if no matches found.	Few or no aspects implemented successfully; limited or no summary displayed; no or limited message provided if no matches found.
Non-functional requirements testing [20]	18+ Testing is organised, structured and exhaustive. 16+ Testing is organised, structured and substantive. 14+ Testing is organised, structured and comprehensive.	A comprehensive black-box test plan that covers most test cases including success, failure, and range.	An acceptable black-box test plan showing individual test cases, but it is not sufficiently extensive.	A limited black-box test plan is evident showing individual test cases, but it is poorly formatted and/or contains few tests.	No evidence of testing provided.
Report [10]	Report critically evaluates the implementation risks, security, and legal considerations related data safeguarding in real world applications.	Report evaluates the implementation risks, security, and legal considerations related data safeguarding in real world applications, evaluation includes facts and opinions of others with appropriate citations.	Report addresses the implementation risks, security, and legal considerations related data safeguarding in real world applications. Evidence is mostly limited to hypothetical situations.	Report addresses some of the implementation risks, security, and legal considerations related to data safeguarding in real world applications. Limited evaluation present.	Report does not address the implementation risks, security, and legal considerations related data safeguarding in real world applications.
Report [10]	The performance of the prototype is documented with an excellent evaluation covering solution applicability, code efficiency, supported with evidence from the test results. Any bugs have plausible solutions suggested. . Report is very well presented, absent of proofing errors, word count +/- 10%.	The performance of the prototype is described with meaningful evaluation covering efficiency, results from testing, and possible bug fixes (if required) but they might not be entirely suitable. Report is well presented, very few proofing mistakes, word count +/- 10%.	The performance of the prototype is described whilst referencing testing results. Bugs are identified but no possible fixes suggested. Report is reasonably presented, minor proofing mistakes, word count +/- 15%.	The performance of the prototype is loosely described but lacks evidence and conviction. Report is reasonably presented, does contain some proofing mistake errors, word count +/- 20%.	Report does not address the prototype performance. Report is poorly formatted, with little meaningful content. Word count +/- 30%.

Deliverables

Your work for this assessment must be submitted in two parts.

PART 1:
This work must be uploaded to the '**Summative 3 Report'** submission portal as a Word document or PDF file. The name of the file should be your student number (e.g. a012345_report). The following sections should be included in your file:

- **Source Code**: this section should contain the entire source code for the prototype. The code should be copied and pasted as per the instructions provided in the 'Guidance' section above and must not be submitted as screenshots.
- **Java Docs**: Add all your Java Docs in this section.
- **Testing**: this section should contain your testing documentation and screenshots of your program outputs.
- **Report**: this section should contain your final report.

PART 2:
This work must be uploaded to the **'Summative 3 Java Code'** submission portal as a .zip file. The name of the file should be your student number (e.g. a012345_code).

You should navigate to the location of your java project, once complete, and zip the folder which contains your project. Please be sure to include the tickets2021.txt and tickets2022.txt in your zipped project.

Please ensure that you have included all of the relevant sections and files in the format specified above, and that you have named your files correctly before submitting to the correct assessment submission portal.

Assessment Criteria
Learning Outcomes: Personal and Transferable Skills
PTS1: Produce software documentation using source code comments PTS2: Prepare reflective report to consider the effectiveness of the solution, workload, planning and development activities
Learning Outcomes: Research, Knowledge and Cognitive Skills
RKC1: Demonstrate knowledge of basic OO concepts including classes, objects, methods and encapsulation, inheritance, abstraction, polymorphism, interfaces RKC2: Build an effective Java solution to a simple requirement specification using appropriate development methods RKC3: Test a software application using a set of documented test cases
Learning Outcomes: Professional Skills
PS2: Describe the legal issues related to software development

Plagiarism
These regulations apply to students registered on a taught programme Teesside University International Study Group (TUISC) course. https://www.tees.ac.uk/docs/index.cfm?folder=Student%20regulations&name=Academic%20Regulations&folder_id=44