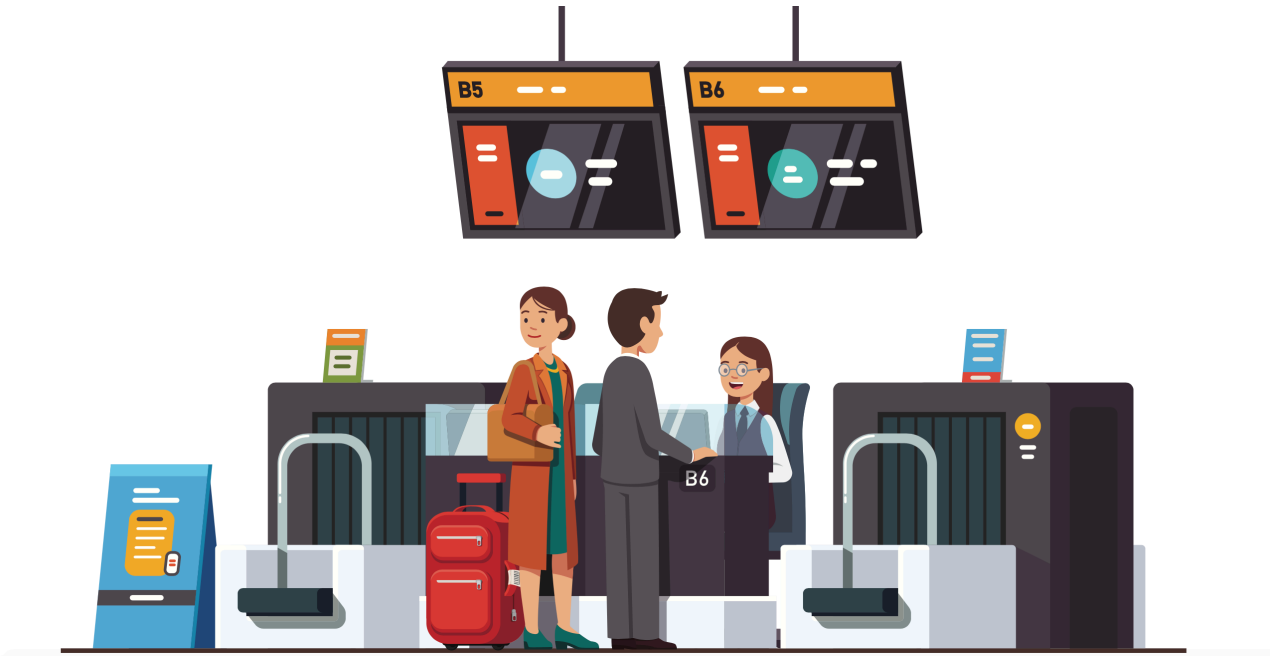
F21AS Assignment

Coursework Report - Check In

Simon Grange, Sara Al Ameri, Mohamed Serry, Bruce Tauro, Rania Ballout
24 January 2020



Introduction

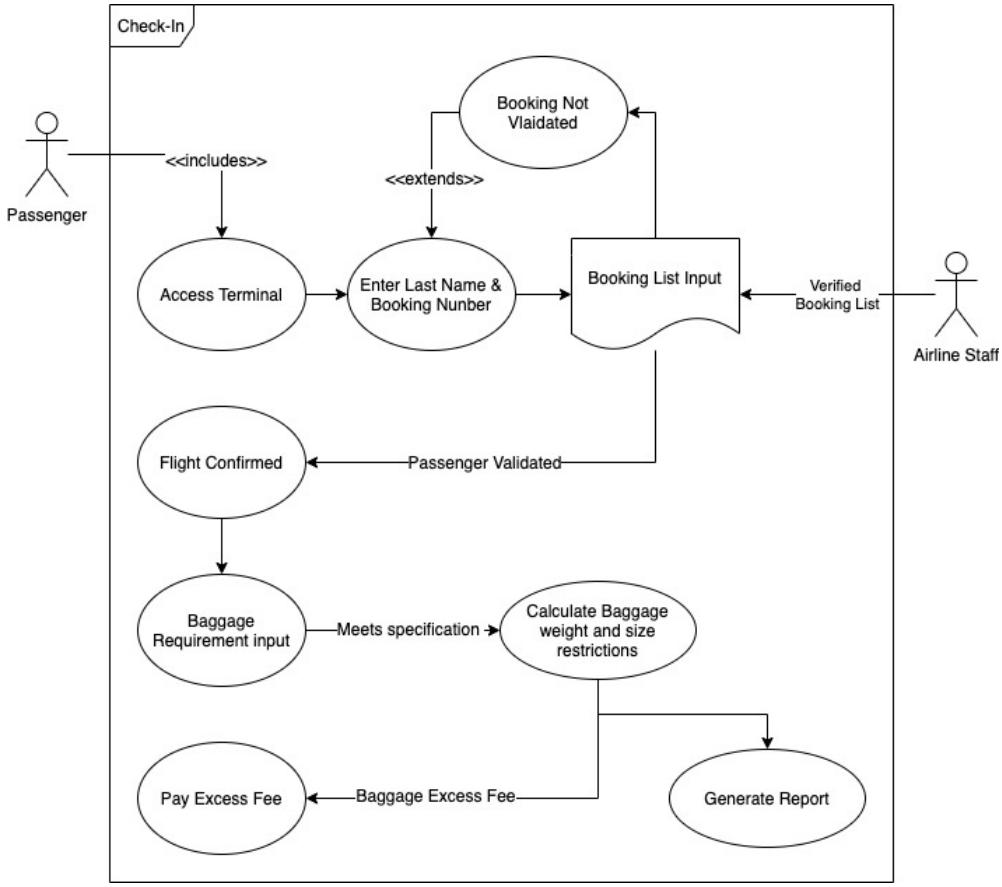
Aa we address the 'pocket airport', exploiting ubiquitous computing and 'dumb' terminals to manage the routine processes in transportation, we need to consider a modular approach which offers passengers a seamless user experience. Let's first focus on the current task of streamlining the check-in processes.

Use case: Check-in

Goal: Passenger checks-in with the system.
Actors: Passenger Check-in
Main Success Scenario:
1.Passenger accesses a terminal
2.Passenger is shown check in GUI
3. Passenger Validation
3a. Passenger enters last name and booking reference,
3b. Baggage dimensions and weight.
4.Generate a report containing:
4a. No of checked-in passengers,
4b. Total weight of baggage checked in,
4c. Total volume of baggage checked in,
4d. Excess fees collected.
Alternate Scenarios:
3a. If the reference is incorrect the passenger cannot be checked in.
3b. If the bag exceeds the weight limits a fee must be paid.

3c. If the bag exceeds the volume limits a fee must be paid.

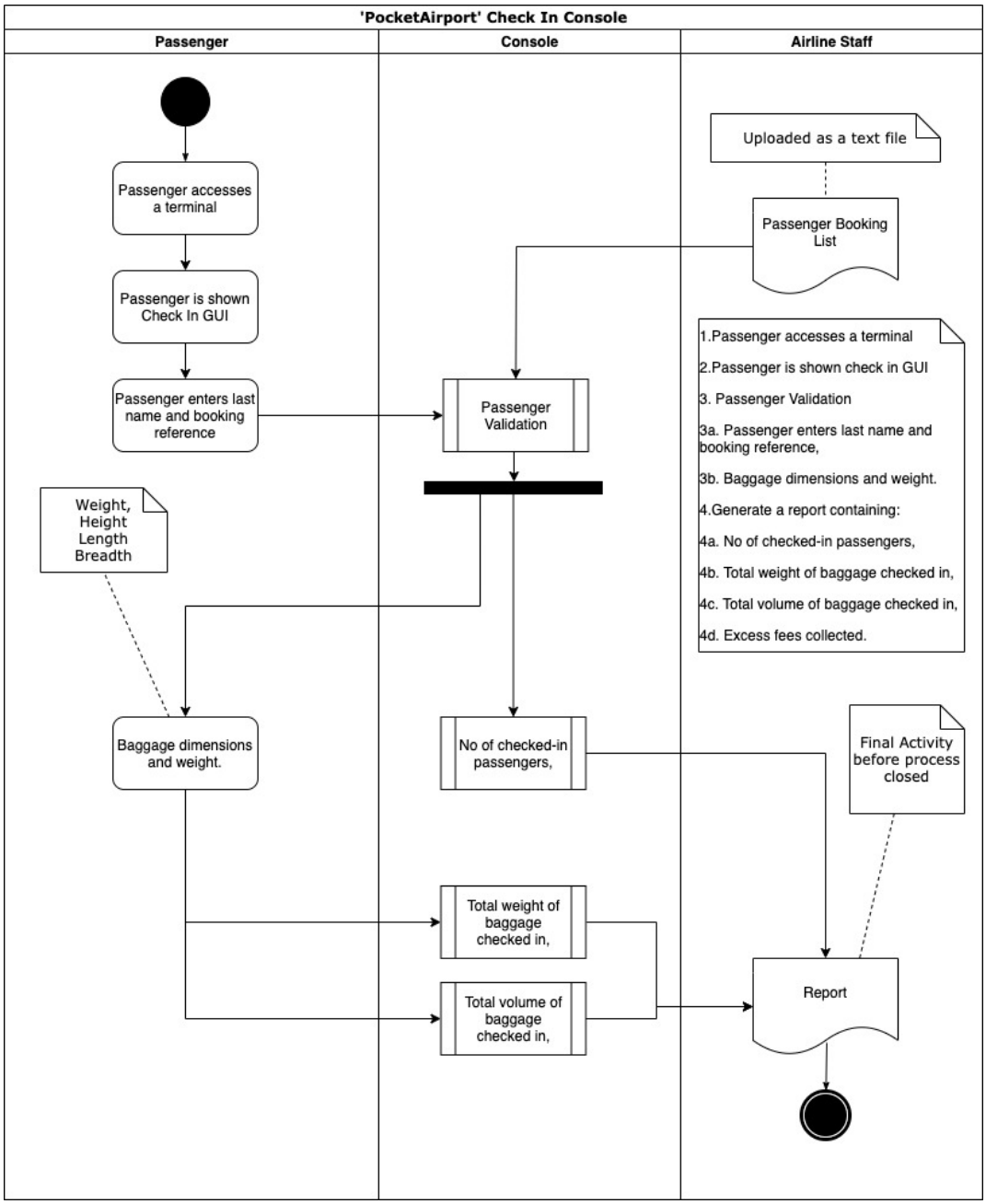
UML Diagrams



Use Case Diagram

Use Case Diagram for 'PocketAirport' CheckIn process

Activity Diagram



Activity Diagram for 'PocketAirport' CheckIn process

Class Diagram

[Please insert class diagram here]

Flight
-code: string -destination: string -airline: string -passengerMAX: int -currPassenger: int -bagWeightMAX: int -currBagWeight: int -bagLMax: int -bagWMax:int -currBagVol: int -excessWeightFee: double -excessVolFee: double -currExcess: double -passengerList: string array
-LoadFlightDb() +calcBaggageHoldVol()

Passengers
-Name: string -Booking: string -Flight: string -checkIn: Boolean
-LoadPassengerDb()

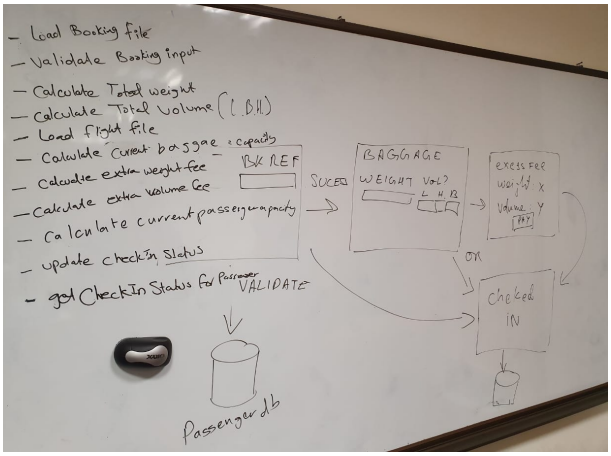
CheckInGUI
-calcExcessWeight() -calcExcessVol() -GenReport()

Our aim is to design and build iteratively and so taking a pragmatic approach to our initially iterative but ultimately agile design methodology, we have set clear goals of reducing the cognitive load not eh passenger at what can sometimes be a hurried and trustful time, and yet where possible ever back to conventional metaphors fr the journey such as the 'timeline', navigation tools that they are likely to be familiar with and paying homage to the traditional paper based systems which most travelers are familiar with in order to reduce the stress of learning new approaches by recognizing an intuitive path.

Conventional Boarding Pass Information given to client to pass to staff and cross check in the system is the starting point for interface design

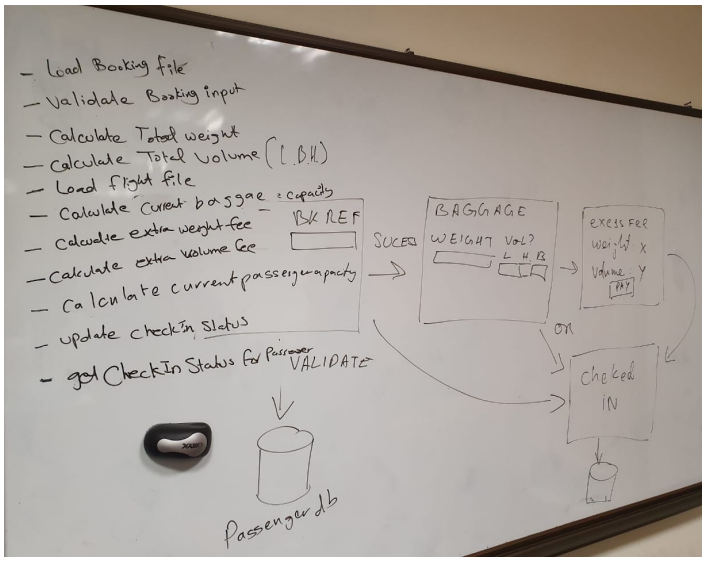
Workshops - Team Meetings

30th January 2020 - HWU C2-20 20.30 - 22.15 All team in attendance



Initial thoughts from workshop 1 - planning the 'PocketAirport'

Identifying the list of functions to perform



Setting the context for UI integration

This emphasizes the need to remain on the narrow path fo the original design brief and avoid over designing at this stage. The consensus opinion was to remain in scope but where practical, design classes that would be adaptable modules for the second stage fo the work, accepting the iterative design process - in accordance with Gall's law; 'Every successful complex system, started with a successful simple system'. An example fo this is the acceptance fo simple I/O text files even though the logical development would be data base management system integration to assist scaling of numbers of passengers. The aim is to ensure the minimal number of screens and transitions and focus n ensuring reliability through testing int he first stage fo development, with a view to automate testing to assist more rapid testing as complexity increases.

Advanced Software Engineering (F21AS)

Coursework 2020

Overview

This coursework contributes 50% of our mark for the course.
The application has been chosen to be complex enough to enable us to try out various software engineering features, whilst small enough to fit in the time available.
The scenario is based around an airport check-in system.
We have defined a group of 5 who are;

First Name	Last Name	Email	Design	Implementation	Report Writing
Mohamed	Serry	Ms366@hw.ac.uk	Threads, Class build and test strategy	Threads, Class build and test	Threads, Class build and test strategy
Safa	Al Ameri	sa299@hw.ac.uk	Client UX (interaction Design), File I/O	Client Screen design, File I/O	Client UX (interaction Design), File I/O
Bruce	Tuaro	Bjt3@hw.ac.uk	User Interface Back End	User Interface Integration and Process logic	User Interface Back End
Rania	Ballout	Rb152@hw.ac.uk	Interfaces and Class Structure	Interfaces and Class Structure	Interfaces and Class Structure
Simon	Grange	sag11@hw.ac.uk	Report Coordinator, Project Space Organiser, User Interface Front End	Report Coordinator, Project Space Organiser, User Interface Front End	Report Coordinator, Project Space Organiser, User Interface Front End

All group members contributed to design, implementation and report writing.
Through internal peer review, each group member was asked to assess the contribution of other members, since if there are significant differences between levels of contribution, then marks will be adjusted accordingly.
The application was developed in two stages:
Stage 1 follows on from materials you will be taught in Part 1 of the course, and concentrates on planning, iterations, data structures, exceptions and unit testing
Stage 2 follows on from materials you will be taught in Part 2 of the course, and focuses on threads, design patterns and agile methods.
The requirements of the application are outlined in the following sections of this coursework specification.
Each stage has two sets of requirements:
Functional requirements, which describe what the application should do.

Software engineering requirements, describing how you should develop the application.

The following deliverables are required:

- A group development plan, handed in before your group begins implementation work.
- Two group reports, describing our group's work, to be written by the whole group.
- A group demonstration, during which our group demonstrates our finished work.

The following **table** shows when each of these deliverables is due, and how much they are worth: Stage Deliverable Weight Deadline Stage 1 Group development plan* 50% 3:30pm, Thursday 6th February (Week 4) Stage 1 Group report 3:30pm, Thursday 27th February (Week 7) Stage 2 Group report 50% 3:30pm, Thursday 26th March (Week 11) Stage 2 Group demonstration Week 12 (times to be announced) *

The development plan will be assessed alongside the Stage 1 group report, so you will not receive separate feedback on this. All deadlines are strict and late submission penalties apply.

F21AS COURSEWORK: STAGE 1

- Stage 1 is designed to assess our understanding of planned iterative development and our knowledge of;
  - Data structures
  - Exception handling
  - Unit testing
  - We are required to develop a simulation of a simple airport check-in system.
  - This first stage develops the basic functionality, which we then extend in Stage 2.

Functional Requirements For Stage 1

- Our application will load in details of existing passenger bookings, show a simple check-in interface, and generate a summary report.
- **1. At the start of the application, all passengers have bought a ticket.** A text file should be provided which gives the list of all bookings, containing, for each passenger, details of the unique booking reference code, the name of the passenger, their flight code, and whether they have checked in or not.
- **2. Another text file is provided showing the details of each flight,** including the destination airport, the carrier, and the capacity of the flight (giving the maximum number of passengers, the maximum baggage weight, and the maximum baggage volume).
- **3. These text files are read at the start of the application, and you can assume that they are correctly formatted *e.g.* the right number of commas in a csv file.** You should check that the booking reference code is correct according to your rules.
- **4. Passengers will need to check in when they reach the airport.** To provide a facility for this, after the list of bookings has been loaded, your application should display a simple GUI representing an electronic check-in kiosk.
- **5. In addition to allowing the passenger to enter their last name and booking reference, this GUI should ask for the dimensions and weight of the passenger's baggage and indicate any excess baggage fee that needs to be paid.** Your group should come up with their own rules for this. You can assume that each customer has only a single piece of baggage.
- **6. The last name and booking reference which are entered into the GUI should be checked against the entries on the flight booking list.** If the details do not correspond, the passenger cannot be checked in, and will have to re-enter their details.
- **7. When the application exits, it should generate a report.** This should indicate, for each flight, the number of passengers checked-in, the total weight of their baggage, the total volume of their baggage, and the total excess baggage fees collected. It should also indicate whether the capacity of the flight is exceeded in any way.

Software Engineering Requirements For Stage 1

- These are the software engineering requirements for Stage 1:
- **1. Your application should be implemented using Java.**
- **2. Develop your program using planned iterative development.** In this stage you should do all the design before writing the code. Decide on all the classes for this stage, their instance variables and methods. Try using CRC cards to help with class design, and use other diagrams where appropriate. Make a plan to divide the work between you, in such a way each person can work independently where possible.

- How will you test your code?
- Decide when and how often you need to meet or be in contact.
- How will you integrate your work?
- **3. Base all your decisions just on the requirements for this stage.** Do not use agile development at this stage, and do not plan ahead to Stage 2. However, do take notes about the development process and your experiences, since you will be asked to summarise this in your report, and may be asked to discuss it in the examination.
- **4. Your program should read the data from the files at the start,** and store it into appropriate data structures. When reading the files, don't think ahead to the GUI or reports; just store the data so that it can be accessed easily (e.g. is a list suitable? would a map be useful?). Then, write methods to analyse the data, which is likely to involve using more data structures. When making your decisions, imagine that you have a large number of passengers and flights.
- **5. Use version control.** Your group should set up a repository, and a link to this repository should be included in your report. Note that we will check the commit history, so make sure this reflects your individual contributions.
- **6. Use exceptions to catch errors in the data.** Each group should decide what makes valid data (e.g., length, range, number of characters, etc.) If an error is found, just continue without that line of data. Provide suitable data to check that your program is working correctly, e.g. input files with some errors.
- **7. You should throw exceptions** in the constructor of at least one class, to ensure that the objects of that class that you create are valid (e.g. booking reference codes), and you should write at least one of your own exception classes.
- **8. Use JUnit to test some of your constructors and/or methods**, particularly ones involving calculations, e.g. to test methods in a Baggage class which includes checks for weights and dimension limits. You could try test-driven development for these methods. If you create a JUnit test for a method, you should test all the paths in the method, not just one.

Development Plan

This is available at;
https://docs.google.com/spreadsheets/d/10vNP79d64IoLKva2wZLCNPJGcB2N2mh4x5TiQ5mDhDQ/edit?usp=sharing

- The report will consist of several sections:
- **1. Names of group members and a summary** (no more than one page) explaining who did which parts of the application and which parts of the report.
- **2. A link to your repository.**
- **3. Does your program meet the specification,** or are there some bits missing or bugs outstanding? Either provide a single sentence "This program meets the specification" or provide details of problems that you know about.
- **4. Suitable UML class diagram(s)** showing the associations between the classes, and the contents of each class.
- **5. Explain which data structures you used,** which classes they are used in, and why you chose them. (Some tools for creating UML diagrams hide the data structure of associated classes. Ensure the reader knows what these are.)
- **6. What decisions has your group made about the functionality** of the program? (This is just information, not a technical report). Include the format of the booking reference codes and details about how excess baggage fees are calculated.

- **7. Testing:** What did you test using JUnit? Give details of which JUnit tests relate to which methods. (No need to print all the code, the marker will look at this electronically). Which types of exception did you use, in which method, and for what purpose?

Submission

- One person in the group should submit this document, which gives details of your class diagram, chosen data structures and work plan, before you start coding. This should be submitted by 3:30pm on Thursday in Week 4.

Submission is mandatory and your group will be penalised if it is not submitted on time. This document should be a maximum of **3 pages** in length.

Group Report - Try to keep the writing concise.

- Your report should be no longer than 12 pages.
- Your development plan, group report and application for Stage 1 will be marked according to the following criteria:
- All reports and code should be submitted through Vision using the links provided in the Assessment section of the course.
- Late submissions will be marked according to the university's late submissions policy, i.e. a 30% deduction if submitted within 5 working days of the deadline and no mark after that.
- If you have mitigating circumstances, please submit the form available at: https://www.hw.ac.uk/students/studies/examinations/mitigating-circumstances.htm Note that all submissions will be checked for plagiarism.
- Criteria Weight
- A (70-100%)
- B (60-69%)
- C (50-59%)
- D (40-49%)
- E/F (<40%)

- Development plan and design decisions 25%
- Clear, well thought out, well justified, and submitted by the deadline
- Mostly clear, thought out and justified, submitted by the deadline
- Some issues with clarity, planning or justification, but submitted by the deadline
- Significant issues with planning or justification, or not submitted on time
- No real indication of planning or thinking, or not submitted
- Functionality 30%
- Functional requirements have been met
- Some requirements not fully met
- A number of requirements are incomplete
- Significant limitations in functionality
- Very little achieved
- Implementation and coding 25%
- Good OOP design, clear modular well commented code, clear class diagrams, appropriate use of version control
- Generally good OOP design and coding with readable class diagrams, appropriate use of version control

- Some issues with OOP design or coding, class diagrams lack clarity, limited use of version control
- Significant issues with OOP design and coding, class diagrams poorly presented or absent, poor use of version control
- Poor OOP design and coding, incomprehensible class diagrams, no use of version
- l control Exception handling and testing 20%
- Effective use of exceptions, thorough unit testing and test data
- Generally good use of exceptions and unit testing with sensible test data
- Some issues with the use of exceptions, unit testing and test data
- Significant limitations with exception handling and testing
- No useful exception handling and testing

F21AS COURSEWORK: STAGE 2

- Stage 2 is designed to assess your command of thread-based programming and design patterns, and your understanding of agile development, which are all taught in the second half of the course.
- In this stage, your application will be extended to simulate passengers queuing at check-in desks and being assigned to waiting planes.
- This will involve the addition of threads and design patterns, and the development of a suitable GUI to show the state of the simulation.

Core Functional Requirements

- These are the core functional requirements.
- Also see the extended requirements section below.
- **1. The simulation consists of multiple check-in desks, a single queue of passengers, and a group of flights waiting to depart.** When passengers arrive at the airport, they join the back of the queue. When they reach the front of the queue, they will be processed by the next available check-in desk, and then assigned to the appropriate flight.
- **2. As for Stage 1, the application will read in details of passengers and flights when it starts up.** Your application should simulate the arrival of passengers at the airport. This can be done by randomly selecting passengers who have not yet checked in. Each passenger has up to one piece of baggage, and the dimensions and weight of the baggage can be chosen randomly when the customer joins the queue.
- **3. After a certain period of time has elapsed, the check-in desks** close and the planes depart. Any passengers remaining in the queue will not be able to board their flights.
- **4. The GUI should show the list of passengers waiting in the queue,** details of what each check-in desk is currently doing (*e.g.* processing a particular passenger, charging for excess baggage), and the status of each flight (*e.g.* number of passengers checked-in, weight of baggage *etc.*) throughout the simulation. One possible example is shown below. Make the timing slow enough that you can watch the displays changing. There's no need to keep the check-in GUI from Stage 1.
- **5. A log should be kept which records events as they happen**, *e.g.* passengers joining the queue, passengers checking in, passengers boarding flights. This log should be written to a file when the application exits.

Software Engineering Requirements

- These are the software engineering requirements for Stage 2:

F21AS COURSEWORK: STAGE 2

- **1. Experiment with agile methods.** For your initial plan, simply decide how many iterations you will have and which features will be developed in each iteration. Then plan each iteration when you start work on it. Plan, design, develop and test each iteration without considering features in the future iterations. You may need to refactor your code at the end of each iteration.
- **2. Spend some of the time trying out agile techniques such as pair programming**, stand-up meetings and time-boxing, and decide whether they are practical for you to use in your project or not. Note down your observations.
- **3. Use threads and ensure that they are synchronized where necessary** and do not interfere with one another. For the core functional requirements, you should have one thread for each of the check-in desks, and one thread to add passengers to the queue. You are encouraged to add more threads to the application when you develop extensions and also consider more advanced scenarios like the producer-consumer model.
- **4. Java provides thread-safe versions of some of its collection classes.** However, you should not use these, since this will limit your ability to demonstrate your knowledge of threading.
- **5. Your design should include design patterns.** As a minimum, you should use the Singleton pattern to implement your log class, and the observer and MVC patterns in your GUI (see the learning materials in Part 2 of the course). You are free to use other design patterns.
- **6. Use packages appropriately.**
- **7. Continue to use version control.**
- **8. The final application should be exported to a jar file, from which the program can be run.** Your group will be required to demonstrate their working program during Week 12.

Extended Functional Requirements

- You are also asked to extend the core requirements. Marks will be awarded based on the complexity of your extension(s) and the knowledge and understanding required to implement them. The following are some suggestions. You are free to come up with your own, though you may want to check with your lecturer before starting work on them:
- **1. Allow the user to alter the speed of simulation using runtime controls.**
- **2. Allow the user to open or close individual desks as the simulation proceed**s, or the simulation could automatically open new desks (up to a limit) when the queue gets to a certain length.
- **3. Your simulation could have more than one queue,** with each queue feeding passengers to more than one check-in desk. You could limit the size of queues. You might introduce different cabin classes (e.g. economy, business) and have queues with different priorities for these.
- **4. Rather than the check-in desks closing at a particular time**, each flight could have a specified departure time, and check-in desks will only accept passengers who arrive before their flight departs. You could show flights departing by removing them from the GUI.
- 5. If you're feeling even more ambitious, you could **simulate more details of the airport experience, e.g. queuing for security.** However, bear in mind that a small number of good quality extensions are better than lots of rushed extensions.

Group Report

- The group report for stage 2 should include the following:
- 1. A brief description of the functionality that your system provides and what it does not do, i.e. does it meet the specification, are there some bits missing or bugs outstanding. How did you extend the core requirements?
- 2. UML class diagram(s) showing the associations between the classes, and the contents of each class – possibly not both in the one diagram.

F21AS COURSEWORK: STAGE 2

3. Details about how we developed your program using agile processes
   - Features we included in each iteration
   - Agile techniques and tools our group used

4. An explanation of how and where threads are used in our application

5. An explanation of how and where design patterns are used in your application

6. Sample screen shots

7. A brief comparison of our development experiences in Stage 1 and Stage 2.

Bruce;
“ “.
Mohamed;
“ “.
Rania;
“ “.
Safa;
“ “.
Simon;
“ “.

Comparison of whether plan-driven or agile development worked better for our group

Problems Encountered

Lessons Learned

Things we would do differently next time are classified according to the stage fo the process;
Design:
Testing:
Implementation:

The technical sections of your report should be written for someone who has studied the material on this course but is not familiar with the coursework.

Diagrams (including UML diagrams where appropriate)

Snippets of code where relevant.

Evaluation Rubrik

First we need to set this int he broader context of the overall course aims, so any extended effort remains inside the scope of the course;

COURSE AIMS

- To consolidate proficiency in imperative programming and software development
- To further develop object oriented programming and object oriented design methods
- To provide knowledge of simple data structures and algorithms
- To introduce concurrent programming techniques
- To instil understanding of the concepts and benefits of advanced software engineering methods
- To give further practical experience of the use of UML in software engineering
- To give practical experience of developing a substantial software engineering team project
- To enable the deployment of patterns in software engineering

LEARNING OUTCOMES – SUBJECT MASTERY

- Skill in the use of UML notation and translation of UML designs to working programs
- Understanding of basic data structures and algorithms and ability to critically evaluate their appropriateness and limitations for a range of moderately complex problems.
- Demonstration of skill in design and implementation of practical GUI based and threaded applications
- To demonstrate a critical understanding of modern software engineering practice and be able to evaluate the strengths and weaknesses of current software engineering methods and techniques - SWOT analysis of our techniques..
- To be able to choose appropriate metrics to measure software quality and quantity in a modern software engineering environment
- To be able to choose a suitable software development environment and development methodology for specific software development tasks and justify the choice

LEARNING OUTCOMES – PERSONAL ABILITIES

- Appreciation of use of methodology to ground system analysis, design and development

- Understanding of different programming paradigms and their inter-relation
- Practice in working in a group, choosing a methodology, reaching a consensus, and working with others to a deadline
- Taking responsibility for own work, taking responsibility in the development of resources, critical reflection on development process and work undertaken by self.
- Effective appreciation of professional standards in modern software engineering practice.
- Showing initiative, creativity and team working skills in collaborative software development

- **SYLLABUS**

Data structures: stacks, queues, lists, priority queues, binary trees

Algorithms: searching (linear and binary) and sorting Advanced object oriented design techniques Thread based programming: thread creation and interaction, shared variables and synchronisation methodologies in software engineering practice;

Unified Modelling Language; design patterns;

Project planning and management in software engineering; Comparison of agile and plan driven approaches

The Report

The report itself will be criticized according tot he following criteria;

- Writing concise?

- Our report should be no longer than 15 pages.

Our group report and application for Stage 2 will be marked according to the following assessment criteria:

By Unknown - http://lccn.loc.gov/50041709, Public Domain, https://commons.wikimedia.org/w/index.php?curid=14901168