4M21 Software Engineering and Design: 2020/2021

Solutions

Version 3

Elena Punskaya

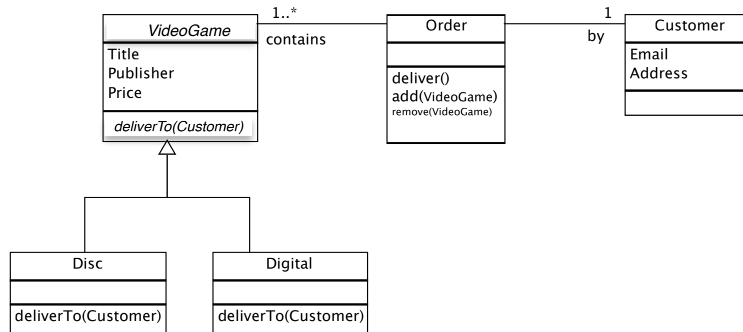
20 May 2021

Q1. **(a)** A purely abstract class that defines only behaviour (not data) is called an interface.

[5%]

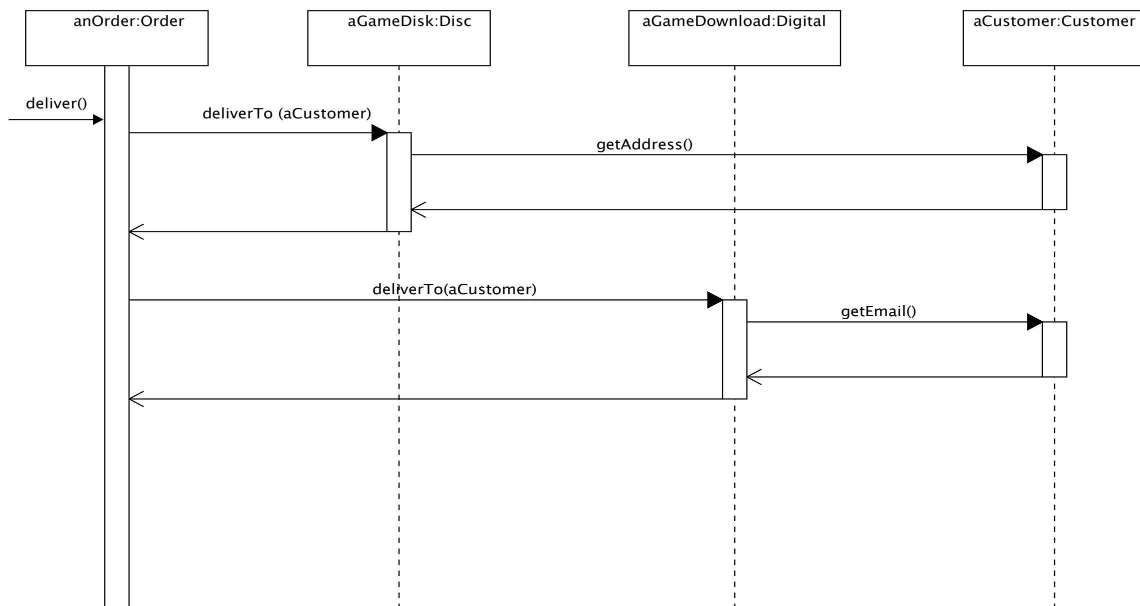
(b) One of possible solutions is presented below.

(i) Class diagram



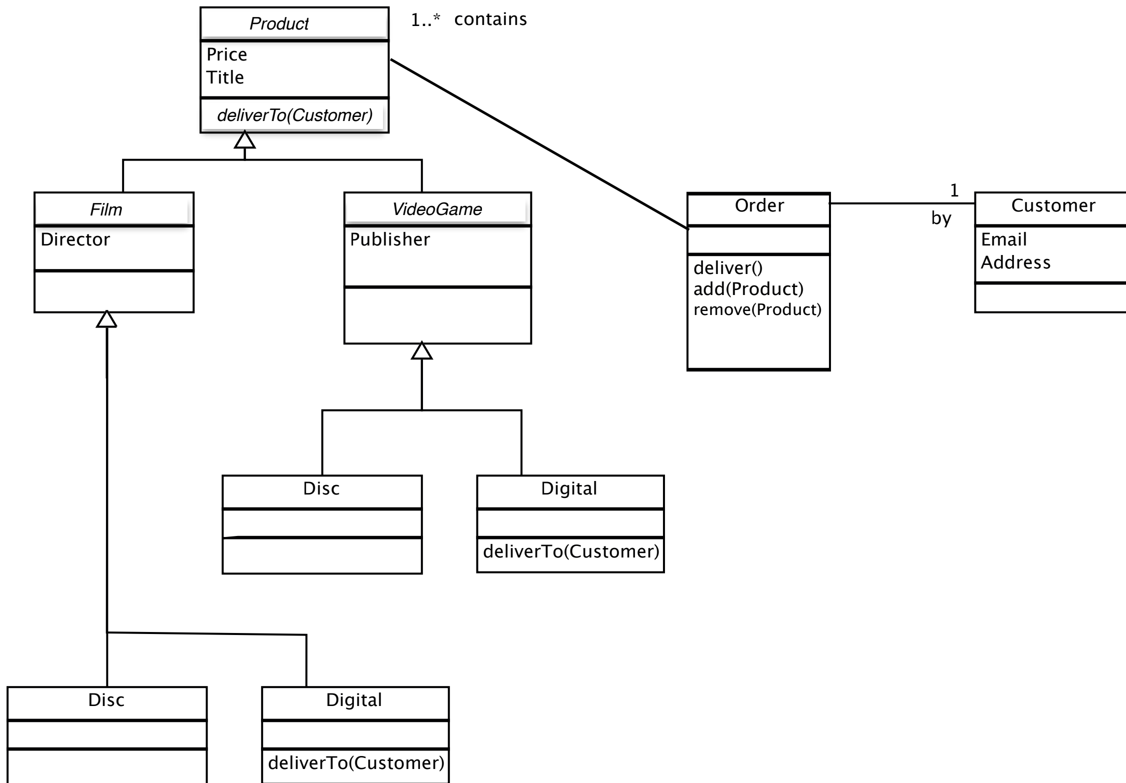
[30%]

(ii) Sequence diagram



[30%]

(iii) Class diagram



[35%]

Assessor's comments: The question was designed to test understanding of the key concepts of the object-oriented design, the ability to interpret the requirements independently, apply the main object-oriented design concepts in practice (in particular, principles of decoupling and abstraction) and the ability to communicate the design through class and sequence diagrams. It was the most popular question and most students were able to go through an independent design process successfully, identifying the key concepts and communicating the outcome clearly using the standard notation. Not everyone was careful when working with both class and sequence diagrams as often sequence diagram did not correspond to the class diagram. Other common pitfalls included inconsistent associations between classes and introducing interfaces without connecting them to the rest of the diagram.

Q2 (a) A use case describes one typical activity from start to end. It is the core of activity based planning and motivates the design, helps with eliciting and organising the requirements and also provides a basis for testing

[5%]

(b) **(i)** Some examples could be: businessman in a rush to have a quick espresso and leave, a coffee lover who particularly appreciates different types of beans, a self-employed consultant working in the café, etc.

[10%]

(ii) User goals: select espresso from the menu and complete the payment in the easiest, most straightforward, quickest way possible

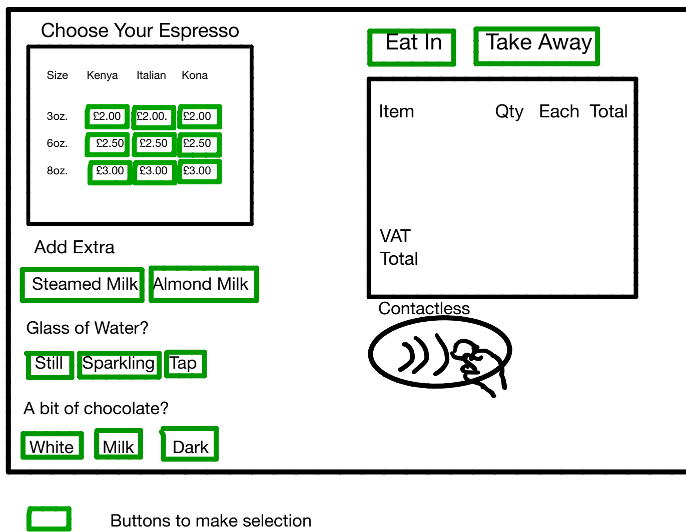
List of pre-conditions: customer has a credit card or contactless payment device (phone, smartwatch)

Criteria for successful completion: the order is placed and the payment completed without any additional help from the staff

Main steps of the activity flow: select type of espresso, select size, add extras such as milk, optionally, add glass of water and a chocolate square. Choose whether to eat in or take away.

Additional/extension: the user might want to be aware of costs prior to making a selection. [25%]

(iii) One of possible solutions is presented below. A Welcome screen and a Success/ Thank you screen could be added to this solution.



Multiple quantities can be entered by pressing the same button several times. [40%]

(iv) Some of the examples (with illustrations of UI implementation) could be:

- displaying waiting time at the end

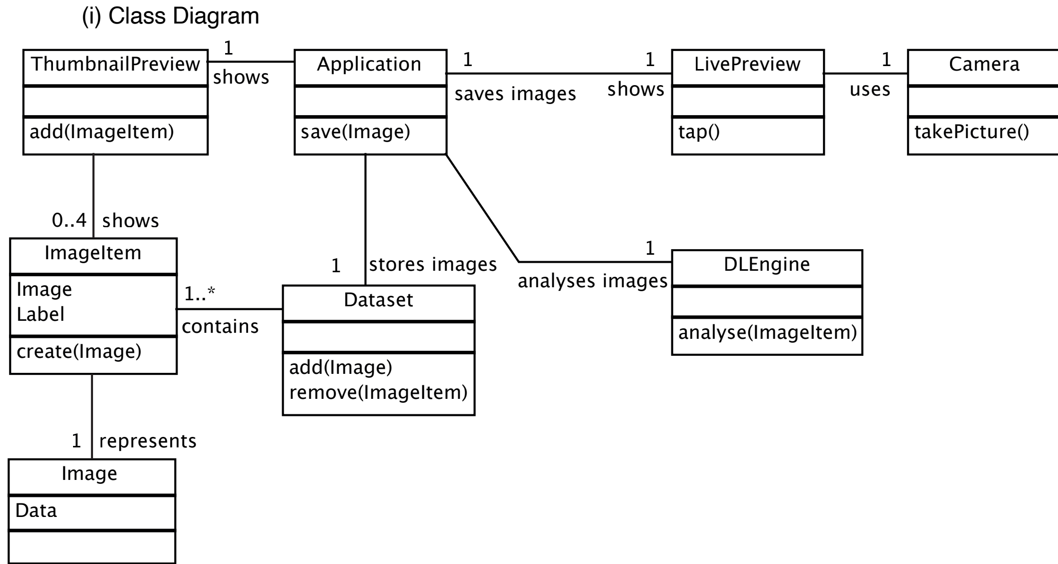
- ask for user feedback [20%]

Assessor's comments: This was a user interface design question that was answered well by most of the candidates. A popular question that candidates were able to complete without any major challenges, however, occasionally some functionality/features were omitted. Some candidates were able to interpret the requirement to streamline the main use case and minimise the number of clicks/limit any additional features; some were able to utilise the relatively large touchscreen making most of the functionality accessible at the same time, and optimised the user experience in the context of non-personal device; however, the majority sketched a very generic design optimised for a small smartphone/smartwatch screen with a number of additional features introduced that are not relevant or even not applicable to the main use case. Some answers to the last part of the question did not focused on "user experience" enhancements for this specific use case – many suggestions involved extending the range of products on offer, introducing extra features that in fact made the process slower or were not applicable in the context of the intended purposes (multi-user Point of Sale system used in-store).

Q3. (a) Decoupling means designing so that the changes to any one unit of the system can be made without affecting another unit of the system. [5%]

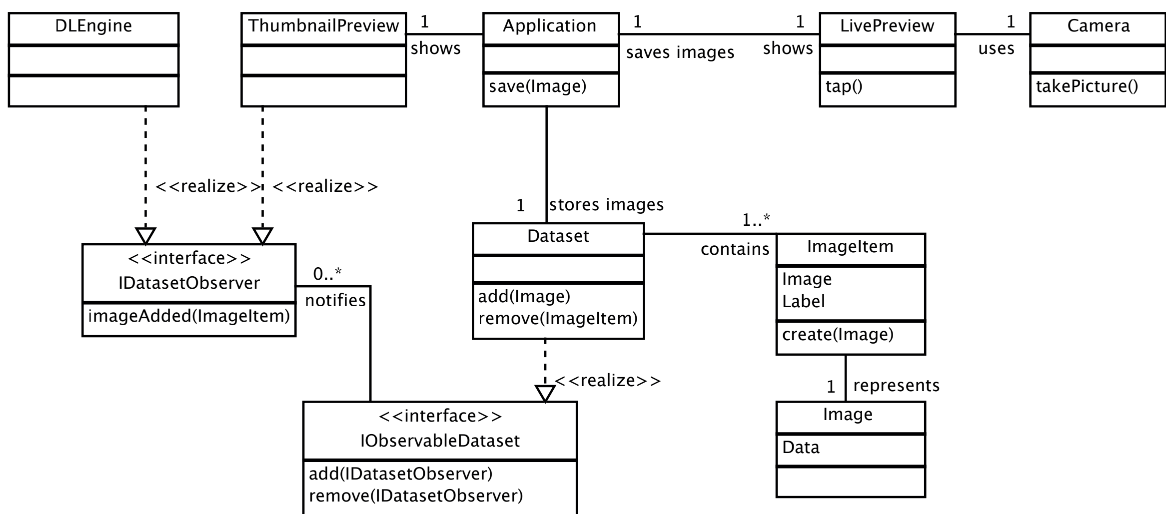
(b) One of possible solutions is presented below.

(i) Class Diagram



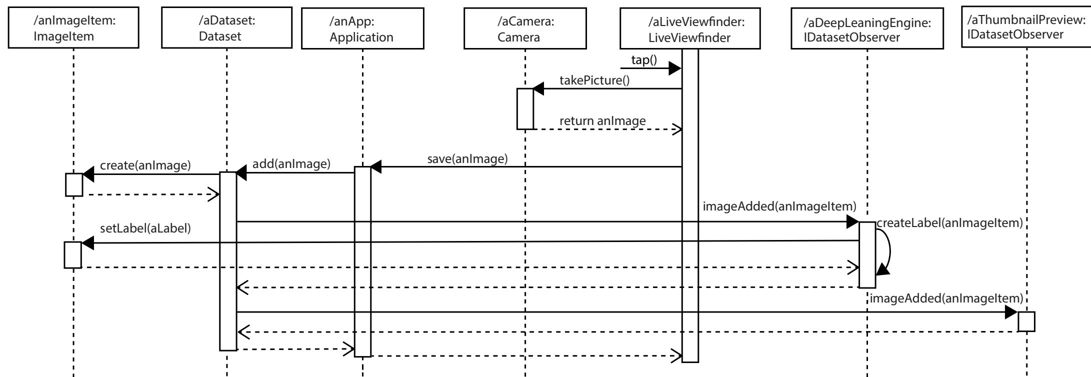
[35%]

(ii) Updated Class Diagram



[30%]

(iii) Sequence Diagram



[30%]

Assessor's comment: The question was on understanding the sequence diagram provided, interpreting the suggested design and extending the design by applying the most appropriate design pattern. The questions was completed reasonably well, and the vast majority of students were able to understand the provided design and extend it using the most suitable design pattern. However, many students found it challenging to derive the class diagrams from the sequence diagram and, in particular, struggled with associations and general consistency between two types of diagrams. Most students were successful with identifying the observer pattern as the one that is most suitable in the last part of the question although rarely were able to apply it to full extend to the specific design.

Q4. (a) The key idea behind Continuous Integration is that integration is unpredictable and has to become a part of each "baby step", i.e. **integrate** and test changes after no more than a couple of hours – check in changes, complete the build and run **entire** test suite before going any further.

The two key practices of CI and its benefits include:

maintaining a single source repository; benefit - avoiding complex merging procedures between a number of branches,

automated build (turning code into a running system**)** launched via a single command line; benefit – saves time and allows to incorporate changes/new features quicker, delivers better reliability and stability, removes the possibility of the human error - going through dialog boxes, moving files, configuration challenges, building different targets, dealing with variety of deployment etc;

Key disadvantages: setting everything up, particularly, given that every commit goes to mainline every day, everything should be optimised to run fast, everything should be included and tested/self-tested is time-consuming, costly and challenging.

[10%]

(b)

(i) Getting the website absolutely right from the beginning is not important over the website lifecycle, and what is "right" today is not necessarily "right" tomorrow, the website is very likely to "evolve". Therefore, a software methodology that reduces cost of change, allows iterations and incremental delivery is most suitable.

As a result, an agile methodology such as extreme programming would be an example of as suitable methodology. Such lightweight approaches are based on the assumption that everything (requirements, business cases, technology, user base among others) changes, and concentrate on improving the software responsiveness to change. They may include a number of different practices with the key enabler being automated testing.

The key practices are: automated builds set up (in particular, ten minute builds), continuous integration, and incremental design, continuous deployment.

The following techniques among others could be used in the project: using "stories" to capture the requirements and implement them incrementally, testing design and requirements via incremental deployments, test-first programming approaches.

Agile project management tools such as weekly cycle and quarterly planning would be complimentary to the approach.

[25%]

(ii) Main risks and appropriate tests to mitigate these risks could be identified by either following the software engineering process or by using "testing quadrants" tools.

Risks identified could include

- implementation and component integration is complex potentially leading to failure of functionality and performance even under ideal conditions - unit tests and integration testing to minimise bugs, automated whenever possible

- poorly understood requirements leading to the system not meeting the acceptance criteria – functional acceptance tests, exploratory/beta testing for certain features etc. written and when possible automated before the development starts and written from the users prospective

- continuous changing requirements leading to functionality/performance failure - automating as many tests as possible, including all unit tests, regression tests, however, also whenever possible UI testing, functional testing, etc.; testing as early as possible ("test-code-refactor" approach) to catch bugs early and optimise to run tests quickly

- failure of functionality/performance in certain real-life conditions (for example, specific browsers) – deployment tests in variety of environments, resource exhaustion/recovery testing

- failure of functionality/performance under load (for example, the website suddenly becomes very popular) - performance testing, stress testing or testing under load

- users might find the system not easily usable resulting in a large number of customer service enquiries / users giving up before they even had a chance to find out all the benefits it provides - usability studies could be introduced

- security and other risks - non-functional acceptance tests

[30%]

(iii) Traditionally, a change to the code may or may not be releasable, release candidate is identified at the end of the process, and testing is significantly delayed, expensive and stressful. Continuous integration formed foundation for Continuous Delivery achieved by constantly running a deployment pipeline that tests whether the software is in the state to be delivered. This is the approach recommended given the risks identified above.

First Stage: developers commit changes into their source repository; the first (commit) stage compiles the code, runs the tests, creates installers; if all ok assemble the executable code into binaries and store them in an artefact repository.

Second Stage: automatically triggered by successful first stage - longer running automated acceptance test (including regression tests)

Third Stage: pipeline branches to enable independent deployment of your builds in various environments (user acceptance tests, capacity tests and production). The testers should be able to see the release candidates available to them and their status plus any comments and at a press of a button deploy in a relevant environment.

Tools: Jenkins (open source),TravisCI, CircleCI, etc.

Keeping the system constantly in production has huge benefits. Some of the advantages include being able to release important features quickly (edge to the competitor), frequent feedback from the users and business colleagues thus the ability to understand the users better and keep changing the system according to users' changing requirements, which would result in a higher chance of success. Automating repeated tasks would save time and resources and would also make system more reliable and stable. The disadvantages however would include investment in automation, more effort required to deploy regularly, automated test and build scripts themselves would become a maintenance overhead and a more close communication between various teams would be required.

[35%]

Assessor's comments: A reasonably straightforward, however less popular, question on software engineering methodologies and their application. Those who did attempt the question answered most parts well. Some candidates did not provide direct answers to the questions and only provided generic statements, for example, not specifically concentrating on "integration" in the first part of the question or "deployment" in the last part of the question, did not justify the reasons for suggesting a specific software development methodology and suggesting a generic testing strategy without concentrating on the risk mitigation.