- Object-oriented analysis

  You are designing an application to help manage recorded GPS (Global Positioning System) data. A location has a latitude and longitude. A track is a named sequence of up to 10000 locations. A waypoint is a named location. A trip is a named collection of possibly track(s), waypoint(s), and other trip(s).

(a) [3] Draw a well-designed UML class diagram to represent these entities in the data model for the application. Provide the correct object-oriented abstractions, relationships, attributes, and multiplicities. State any further assumptions, and highlight their appearance in the diagram.

(b) [2] Accordingly, write correct skeletal Java code for these entities in the data model of the application. Include all abstractions, relationships, attributes, and basic public methods. For example, named entities should have a public `getName` method.

- Object-oriented design

[1] Suppose a system event log contains events, with just the need to append and iterate over events. A potential software design has corresponding `Log` and `Event` classes. A developer decides to have `Log` inherit from `ArrayList<Event>`. Give two different reasons that clearly explain why this may not be an appropriate software design decision. (Giving an alternative design by itself is not a reason.)

1:

2:

- Requirements

[2] For a mobile application to help a claimant note expenses for a travel expense claim report, give an example user story for a user requirement and an example user story for a non-functional requirement. For each user story, also provide two acceptance tests for the requirement.

user requirement user story and two acceptance tests:

non-functional requirement user story and two acceptance tests:

• Requirements

Consider a task with three required subtasks: pre-authorizing payment, fueling at a gas pump at a service station, and settling payment. The pre-authorizing payment subtask has two variations: pre-authorizing by credit card or pre-authorizing by debit card. Fueling is allowed once payment has been pre-authorized.

(a) [1] Draw the correct UML use case diagram for the task, subtasks, and task variations, showing the actor(s), use cases, and relationships.

(b) [2] Write a correct use case description for the fueling subtask, with the following fields. List the steps of what the actor(s) do and what the system presents in the basic flow.

use case name:

participating actor(s):

goal:

trigger:

precondition:

postcondition:

basic flow:

…

• [3] Requirements

Consider the behavior of a bounded stack that can contain at most three items. There are the usual operations push, pop, and top to, respectively:
• push an item onto the top of the stack (if not full),
• pop the top item off the stack (if not empty), and
• get the top item on the stack (if not empty)

Draw a correct UML state diagram to express the states of this stack with the usual behavior resulting from the operations. The initial state is an empty stack.

• [3]		Testing

Consider a `Rect` class to represent a rectangle in a two-dimensional plane.

```
public class Rect {
    …
    // create rectangle with given corners
    public Rect( Point topLeft, Point bottomRight ) { … }

    // return true iff point p is in or on the rectangle
    public boolean encloses( Point p ) { … }
}
```

Draw a thorough set of equivalence classes of tests for the `encloses()` method., indicating the expected results. Add explanatory text as needed for analogous or unusual cases. You do not need to implement the method, write test code, or use JUnit. State your assumptions.

• [2] Software process

What is not true of the *waterfall* software process model?

(a) The process is simple and readily understood.
(b) Software requirements can be changed easily later.
(c) The customer sees working software early on.
(d) The phases in the process are completed one after another.

Correct choice(s):

In *incremental prototyping*, the software product is built up by adding successive increments. Which one of the following best describes what gets done in these increments?

(a) Basic forms of features get done first, then refined variations get done next.
(b) Must do features get done first, then could do features get done next.
(c) Defects are fixed first, then new features get done next.
(d) Must do features get done first, then should do features get done next.

Correct choice:

Which one of the following best upholds the Extreme Programming practice of *simple design*?

(a) Give your software product a simple, easily remembered name.
(b) Design just what you need to make your high-priority requirements work.
(c) Create a design that covers many future possibilities.
(d) Make detailed designs of all your requirements.

Correct choice:

Which one of the following helps to *validate* a software product?

(a) Write tests first to express the requirements in some executable way.
(b) Analyze the product code using a model checker to look for potential faults.
(c) Have developers conduct code reviews as they implement the product.
(d) Release prototypes to customers to receive feedback on meeting their needs.

Correct choice: