- Object-oriented analysis and design

  (Learning goal: given problem description, draw correct UML class diagram; apply object-oriented design principles.)

  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).

[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.

- Object-oriented analysis and design

  (Learning goal: from/given UML class diagram, write correct Java code)

[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 analysis and design

  (Learning goal: given Java code, draw correct UML class diagram.)

[3] Draw the corresponding UML class diagram for the following skeletal Java code. Provide the correct abstractions, relationships, attributes, methods, and multiplicities.

```
class A {
    private int i;
    public int get() { … }
}

interface B {
    public int set();
}

class C {
    public float f;
    public float get() { … }
}
```

```
class D {
    public String get() { … }
}


public class E extends A implements B {
    private ArrayList<C> c;
    private D d1, d2;

    public int set() { … }
}
```

- Object-oriented analysis and design

  (Learning goal: explain whether inheritance is or is not appropriate.)

[2] 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:

- Object-oriented analysis and design

  (Learning goal: explain the differences between generalization with inheritance and interfaces.)

[2] Consider abstract classes and interfaces in Java. Explain clearly in what situation(s) should abstract classes be used and in what situations should interfaces be used.

• Object-oriented analysis and design

(Learning goal: explain coupling and cohesion and their relationship.)

[2] Consider coupling and cohesion for the classes of an object-oriented application. Explain clearly how and why these concepts are related. (Defining the terms is not enough.)

- Software process

  (Learning goal: explain and relate agile manifesto, agile principles, and agile practices.)

[2] The Agile Manifesto describes a principle: “Welcome changing requirements, even late in development.”

From specific Extreme Programming practices, describe two that help to achieve this principle. Explain clearly how. (Just restating the principle or defining the practices is not enough.)

[2] The Agile Manifesto states as a principle: “Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.” From specific Scrum practices, describe two that help to achieve this principle. Explain clearly how. (Just restating the principle or defining the practices is not enough.)

- Software Process

  (Learning goal: explain and give examples of the differences between incremental and evolutionary prototyping.)

[2] Consider the versions of an operating system that you know. For that system, give a clear example of incremental prototyping and evolutionary prototyping. For each example, explain clearly why it applies a certain type of prototyping. (Do not give examples from the course.)

incremental:

evolutionary:

- Requirements

  (Learning goals: given problem description, determine a relevant user story; distinguish and give examples of requirement types; for a requirement, provide acceptance tests.)

[4] 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

  (Learning goal: given problem description, draw correct UML use case diagram)

  Consider a task with two required subtasks: pre-authorizing payment and fueling at a gas pump at a service station. The pre-authorizing payment subtask has two variations: pre-authorizing by credit card or pre-authorizing by debit card.

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

- Requirements

  (Learning goal: from/given task, write correct use case description.)

[3] Accordingly, 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:

…

- Requirements

  (Learning goal: given behavior description, draw correct UML state diagram.)

[3] Consider the behavior of a payment machine that accepts only loonies (one dollar coins) and toonies (two dollar coins), one-at-a-time. At least three dollars need to be inserted for the machine to automatically display a confirmation number. The user can also eject the coins inserted so far, if under three dollars. If over three dollars were inserted, the extra amount is returned as change.

For this behavior, draw a correct UML state diagram. Include the relevant states, transitions, triggers, guards, and actions. State any further assumptions, and highlight where each appears in the diagram.

• Testing

[3] Consider a `Rect` class to represent a rectangle in a two-dimensional integer plane.

(Learning goal: given a programmer interface, determine equivalence classes of tests to verify its required behavior.)

```
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 ) { … }
}
```

Depict a thorough set of test case equivalence classes for the `encloses()` method., also indicating the expected results. Add explanatory text as needed for analogous or unusual cases. State your assumptions. Do not implement the method. Do not write test code. Do not use JUnit.