Introduction to Software Engineering (SS 2018)

Final Exam Solution

Grading Scheme:

Grade Intervall		Grade
0	17.5	5.0
18	20.5	4.7
21	23.5	4.3
24	26.5	4.0
27	29.5	3.7
30	32.5	3.3
33	35.5	3.0
36	38.5	2.7
39	41.5	2.3
42	44.5	2.0
45	47.5	1.7
48	50.5	1.3
51	60	1.0

Question 1 — Multiple Choice [12 points]

a) [1p] A UML communication diagram describes:

☒ **The static and dynamic behavior of a system.**
☐ Only the static structure of a system.
☐ Only the functionality of a system.
☐ The functionality and the structure of a system.
☐ Only the dynamic behavior of a system.

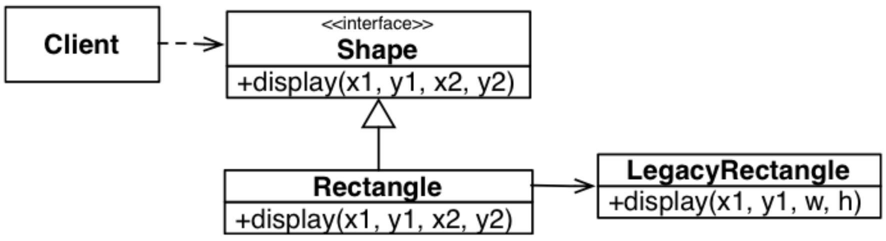
b) [1p] Which of the following statements is correct?

☒ **A software architecture is an instance of an architectural style.**
☐ An architectural style is an instance of a software architecture.
☐ A subsystem decomposition is a pattern for an architectural style.
☐ A software architecture is a pattern for a subsystem decomposition.
☐ A subsystem decomposition is a pattern for a software architecture.

c) [1p] Which of the following statements is correct?

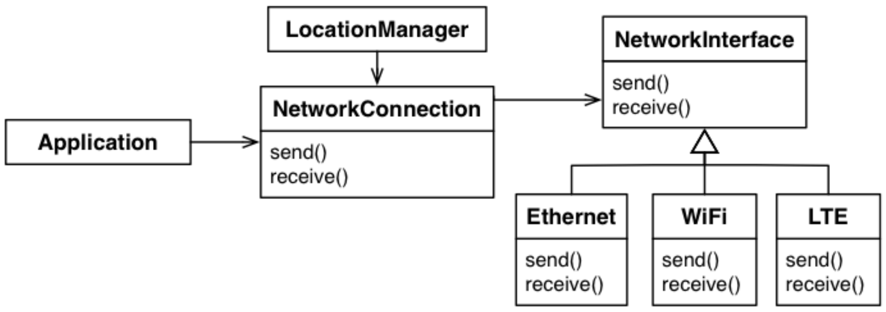
☒ **To promote local changes, a developer need to stage, commit and push.**
☐ To download remote changes, a developer needs to push.
☐ Merge conflicts are resolved on the remote repository.
☐ Pull is a compound command, composed of fetch and clone.
☐ Fetching changes can be done offline.

d) [1p] Which pattern is used in the following UML diagram?



☐ Strategy Pattern
☒ **Adapter Pattern**
☐ Composite Pattern
☐ Bridge Pattern
☐ State Pattern

e) [1p] Which pattern is used in the following UML diagram?



☒ **Strategy Pattern**
☐ Adapter Pattern
☐ Composite Pattern
☐ Bridge Pattern
☐ State Pattern

f) [1p] A tutor and a student discuss the homework solution on Slack. Which of the following aspects describe this communication event?

- [✓ **Unscheduled, asynchronous and informal**]
- ☐ Unscheduled, synchronous and formal
- ☐ Scheduled, asynchronous and informal
- ☐ Scheduled, synchronous and formal
- ☐ Unscheduled, asynchronous and formal

g) [1p] What is a difference between defined process control and empirical process control?

- [✓ **In defined process control, every process step must be completely understood, which is not the case in empirical process control.**]
- ☐ Empirical process control sees deviations as errors, while defined process control sees them as opportunities.
- ☐ Defined process control deals better with changes than empirical process control.
- ☐ Given a well defined set of inputs, empirical processes create the same output every time, while the application of defined process control can lead to different output.
- ☐ Empirical process control cannot deal with changes.

h) [1p] Which of the following statements is correct?

- [✓ **V-Model is linear.**]
- ☐ Unified Process is entity-oriented.
- ☐ Spiral Model is sequential.
- ☐ Waterfall Model deals well with changes.
- ☐ Scrum is sequential.

i) [1p] Which of the following statements is correct?

- ☐ Generalization is only found during the implementation of the system.
- ☐ Specialization is the same as delegation.
- ☐ Specialization means to first determine the subclass and then the superclass.
- [✓ **Generalization means to first determine the subclass and then the superclass.**]
- ☐ Generalization is the same as delegation.

j) [1p] Which of the following statements is correct?

- ☐ Refactoring is an example of adaptive development.
- ☐ Iterative development means to react to changing requirements.
- ☐ Iterative development adds new features to existing software versions.
- [✓ **Adaptive development means to react to changing requirements.**]
- ☐ Incremental development means to react to changing requirements.

k) [1p] Which activity is <u>not</u> part of a typical change control process?

- ☐ Evaluate change request
- ☐ Review change request
- ☐ Audit change
- ☐ Identify change
- [✓ **Define controlled item**]

l) [1p] Which of the following statements is correct?

- ☐ Authority is the commitment of a role to achieve specific results.
- ☐ Accountability is the ability to make binding decisions between persons and roles.
- ☐ Responsibility is tracking a task performance to a specific person.
- ☐ Delegation is transferring accountability to another person.
- [✓ **None of the mentioned options.**]

Question 2 — Requirements Analysis [11 points]

a) **[3p]** Identify requirements in the problem statement shown below. Define the type of requirement for each of them. For non-functional requirements / constraints, also define the category (e.g. supportability). Identify at least two functional and four non-functional requirements / constraints.

Problem Statement: *"I want an application to organize tournaments of computer games. Players should be able to participate using a low bandwidth connection. The software should be easily extensible to support new game types and there should be an online help for every game. The application has to use the existing database server. I want the software delivered on an encrypted USB stick. It has to be written in Java and must only use open source components."*

Solution:

Functional requirements:
- organize tournaments of computer games
- an online help for every game

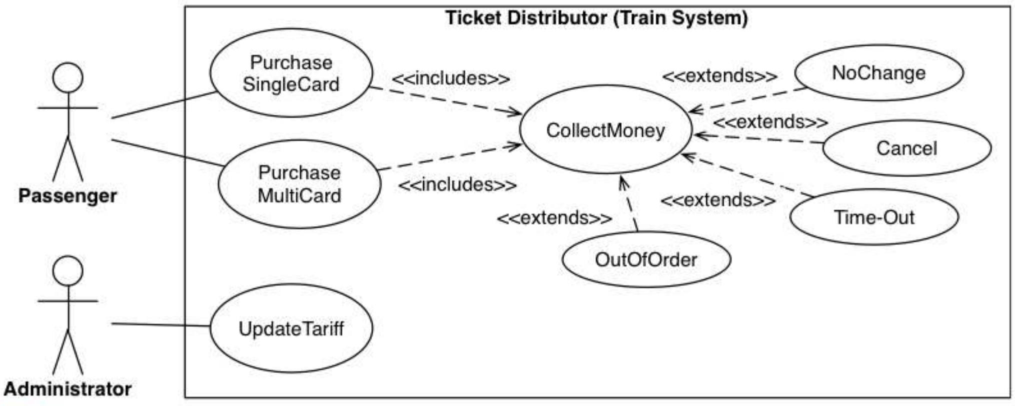
Non-functional requirements:
- Performance - Throughput: low bandwidth connection
- Supportability - Extensibility/Adaptability: extensible to support new game types

Constraints:
- Packaging: delivered on an encrypted USB stick
- Implementation: written in Java
- Interface: only use open source components
- Interface: existing database server

Correction Criteria:
- + 0.5 points for each correct functional requirement
- + 0.5 points for each correct non-functional requirement including the correct category

b) **[4p]** Draw a UML use case diagram for a ticket distributor of a train system. The system includes two actors: a passenger, who purchases different types of tickets, and an administrator, who maintains a reference database for the tariff. The diagram should include: `PurchaseMultiCard`, `PurchaseSingleTicket`, `CollectMoney`, `UpdateTariff`. Also model the following exceptions when collecting money: `Time-Out` (i.e. the passenger took too long to insert the right amount), `Cancel` (i.e. the passenger selected the cancel button without completing the transaction), `OutOfOrder`, and `NoChange`.

Solution:



Correction Criteria:
- + 0.5 points for each correctly modeled use case including all corresponding associations and the connection to the correct actor
- - 0.5 points for mistakes
- 0 points for the wrong notation or the wrong UML diagram type

c) **[4p]** Provide a textual description of the use case `CollectMoney` by filling out the following table.
Hints: When you describe the event flow, make sure to use indentation to distinguish between the actor and the system steps.

Solution:

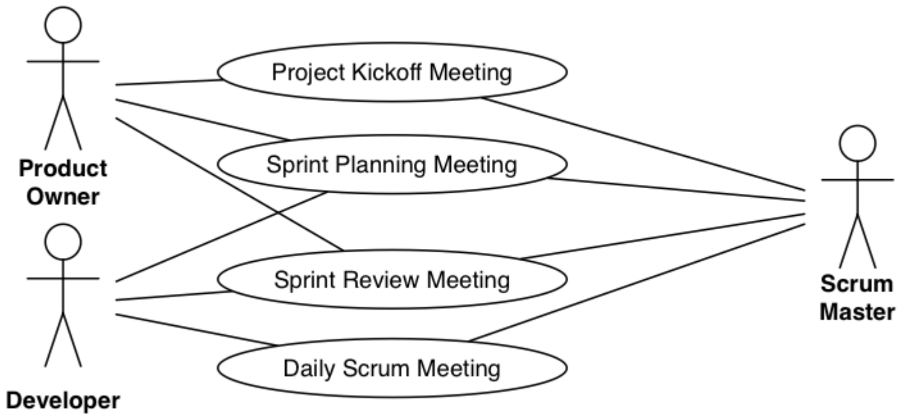
Use case name	CollectMoney
Participating Actors	Initiated by Passenger
Flow of events	1. The Passenger clicks on "Buy ticket" button.
	2. The TicketDistributor asks the Passenger to insert the required amount of money.
	3. The Passenger inserts cash into the ticket distributer.
	4. The TicketDistributor counts the money and displays the remaining amount to be paid (because the money is not enough).
	5. The Passenger inserts cash into the ticket distributer.
	6. The TicketDistributor counts the money (now it is enough), releases the Passenger's change (if there is any), prints and issues the ticket.
	7. The Passenger takes the change and the ticket.
Entry condition	The Passenger has chosen a ticket type The TicketDistributor has change The TicketDistributor has enough paper to print tickets The TicketDistributor has enough ink to print tickets
Exit condition	The TicketDistributor has more money than before The ticket is valid for the chosen journey The TicketDistributor released the correct amount of change
Special requirements	It takes less than one minute to collect the money

Correction Criteria:
- + 0.5 points per correct step in the flow of events (up to max. 2 points)
- + 0.5 points for filling out each of the other sections correctly

Question 3 — Modeling Scrum [13 points]

a) **[3p]** Create a functional model for the Scrum activities with UML. **Hint:** Include the Scrum roles as actors.

Solution:

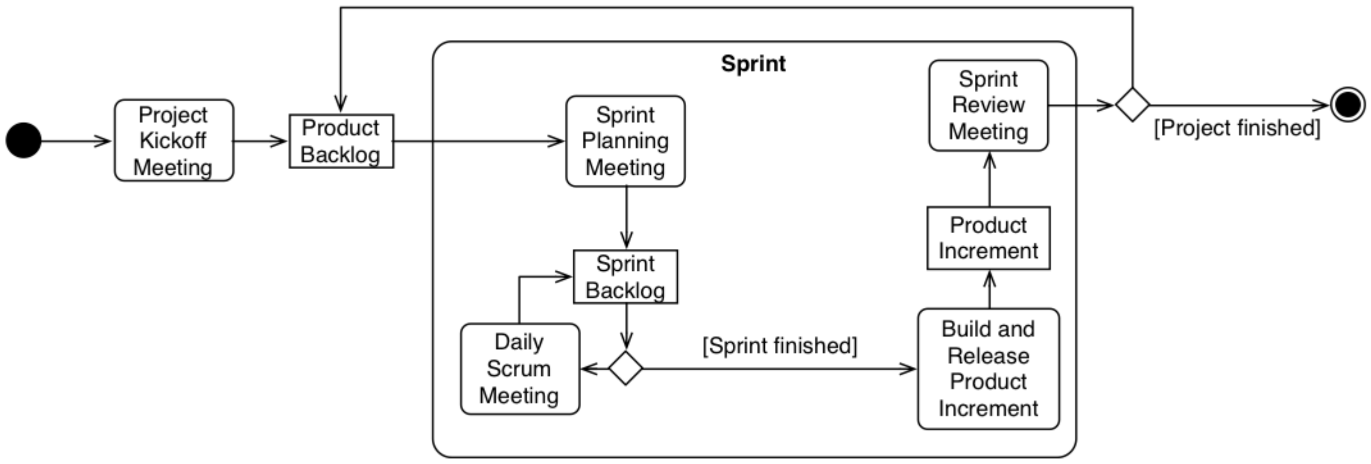


Correction Criteria:

- + 0.5 points for each Scrum activity modeled as use case
- + 0.5 points for each Scrum role modeled as actor
- + 0.5 points if all connections for one actor are correct (per actor)
- - 0.5 points for each wrong role
- - 0.5 points if at least one connection of an actor is wrong (per actor)
- 0 points for the wrong notation or the wrong UML diagram type

b) **[5p]** Create a dynamic model of the Scrum process with a UML activity diagram

Solution:

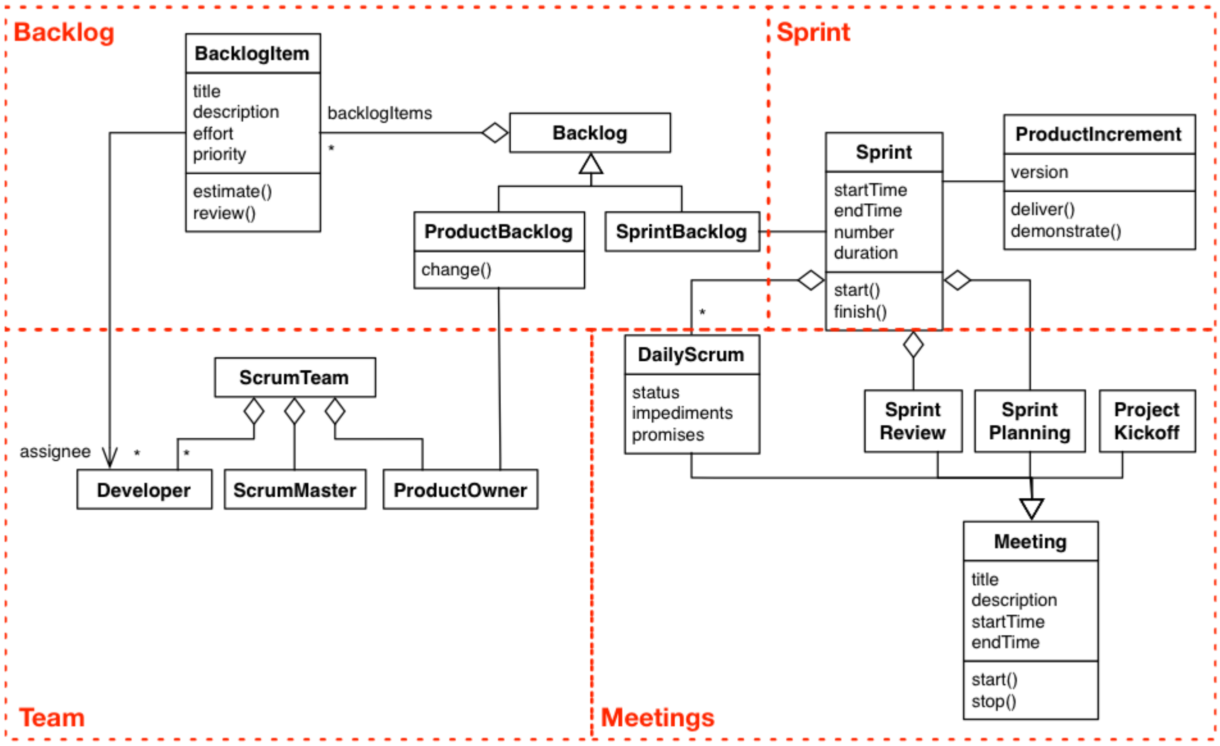


Correction Criteria:

- + 0.5 points for each correctly modeled core activity and core artifact in the right order (up to 4 points)
- + 0.5 points for including a start and end node
- + 0.5 points for each correct decision node
- 0 points for the wrong notation or the wrong UML diagram type

c) **[5p]** Create an analysis object model for Scrum. **Hints:** Focus on the core concepts of Scrum. Include attributes and methods in your model.

Solution:



Correction Criteria:

- + 1 point for each dotted area “Backlog”, “Sprint”, “Team” and “Meetings”, if they contain at least one class, attribute and method
- + 1 point for at least 3 correct associations between these areas
- - 0.5 points for every mistake
- 0 points for the wrong notation or the wrong UML diagram type

Question 4 — System Design [6 points]

a) **[2p]** Explain the similarities and differences between the **pull notification variant** and the **push notification variant** in the Model View Controller architectural style.

Solution:

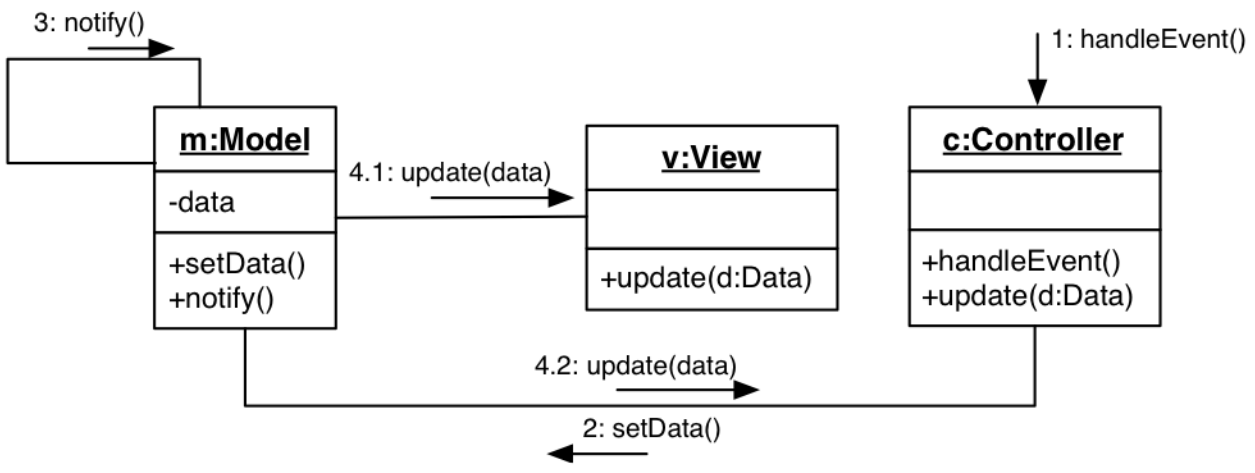
Similarities:
- In both variants, View and Controller are observers of the Model
- In both variants, the Controller has an association to the Model
- In both variants, the Model is associated with a set of Observers

Differences:
- Pull:
  - Both, View and Controller have an association to Model and call getData()
  - The notify() method calls update() without data
- Push:
  - Only the Controller has an association to Model
  - The notify() method calls update(d: data) with data

Correction Criteria:
- + 1 point for each well explained similarity / difference

b) **[4p]** Create a UML communication diagram for the **push notification variant** of the Model View Controller architectural style.

Solution:



Correction Criteria:
- + 1 point for each correct message (when the method for the message exists in the target object and the arrow has the right direction)
- - 0.5 points if the objects are not instantiated, i.e. underlined
- 0 points for the wrong UML diagram type or the wrong sequence of messages

Question 5 — Object Design [11 points]

a) **[3p]** Explain similarities and differences between the **strategy pattern** and the **bridge pattern**. Provide examples when to use which pattern.

Solution:

- Similarities:
  - Both use first delegation, then inheritance
  - Both allow the exchange of the delegation target at runtime (bridge pattern only at startup, strategy pattern anytime)
- Differences:
  - The Bridge Pattern encapsulates an abstraction from its implementations, the Strategy Pattern encapsulates an algorithm from its implementations
  - The Strategy Pattern has a Policy class that decides which ConcreteStrategy to instantiate, the Bridge Pattern has a hard-coded decision in the constructor which subclass to instantiate
  - The Bridge Pattern delays the binding of interface and sub-class to the start-up time of the system; the Strategy Pattern allows to switch between specific implementations at run time

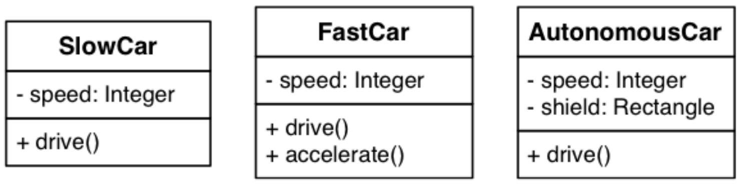
Examples:

- Strategy Pattern: allowing to change the concrete implementation of a sort algorithm at runtime
- Bridge Pattern: setting the encryption length of the cipher at startup time

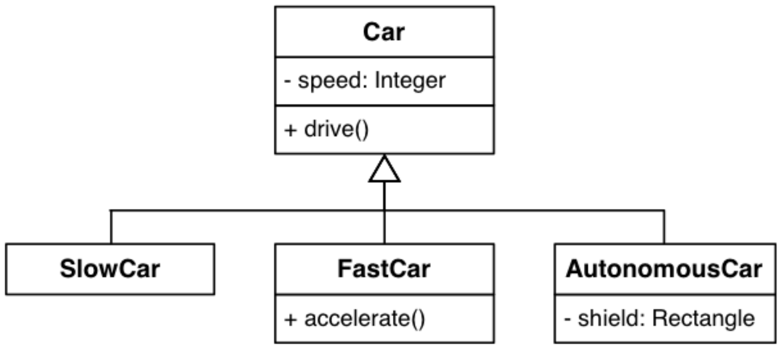
Correction Criteria:

- + 0.5 points for each valid and well explained similarity / difference (max 2 points in total)
- + 1 point for each good example
- 0 points for definitions from the slides

b) **[3p]** Propose a model refactoring for the following object model and justify your solution. Create a new UML diagram with the refactored solution.



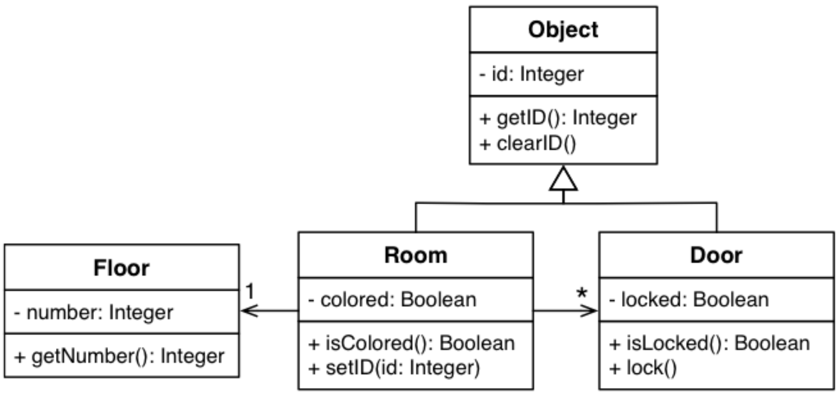
Solution:
Move all common attributes and methods into a new super class Car to avoid duplicated code and improve the maintainability and extensibility of the object design.



Correction Criteria:

- + 1 point for the correct model refactoring including a good justification
- + 1 point for creating the super class Car with the correct inheritance relationship
- + 0.5 points for moving the speed attribute to Car
- + 0.5 points for moving the drive() method to Car

c) **[5p]** Map the following UML class diagram to Java code. **Hints:** Minor syntax errors are OK (e.g. missing semicolons). Include everything shown in the diagram including methods that return and change associated objects. Constructors are not needed.



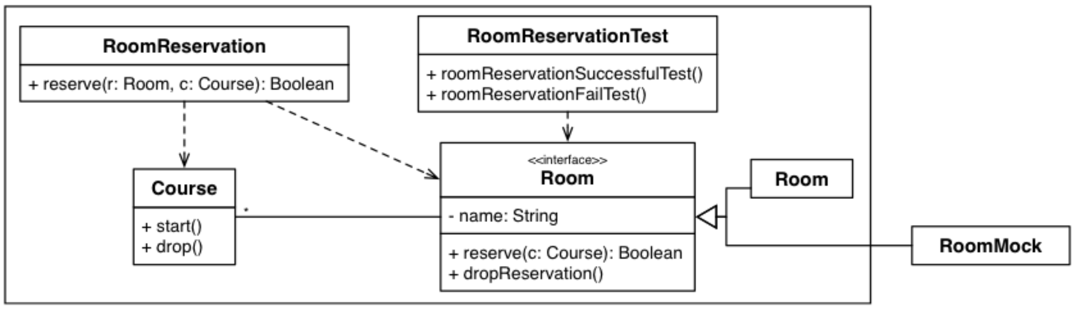
Solution:

```
public class Object {
        private Integer id;
        public Integer getID() {
                return id;
        }

        public void clearId() {
                id = null;
        }
}

public class Room extends Object {
        private Boolean colored;
        private Floor floor;
        private Set<Door> doors = new HashSet<Door>();

        public Boolean isColored() {
                return colored;
        }

        public void setID(Integer id) {
                this.id=id;
        }
}

public class Door extends Object {
        private Boolean locked;
        public Boolean isLocked() {
                return locked;
        }

        public void lock() {
                locked = true;
        }
}

public class Floor {
        private Integer number;
        public Integer getNumber() {
                return number;
        }
}
```

Correction Criteria:
- + 1 point for each correctly implemented class (including attributes, methods, inheritance)
- + 0.5 points for adding the line "private Set<Door> doors" to the Room class
- + 0.5 points for adding the line "private Floor floor" to the Room class
- - 0.5 points for mistakes

Question 6 — Testing [7 points]

Consider the following object design model for a university application with a room reservation service.


RoomReservation
+ reserve(r: Room, c: Course): Boolean
RoomReservationTest
+ roomReservationSuccessfulTest()
+ roomReservationFailTest()
Course
+ start()
+ drop()
*
<<interface>>
Room
- name: String
+ reserve(c: Course): Boolean
+ dropReservation()
Room
RoomMock

a) **[4p]** Below you can see an incomplete unit test for reserving a room where parts of the test code are missing. Insert the missing parts of the code at the correct position, based on the given UML diagram. The current system under test (SUT) misses the collaborating object Room. To implement the test, create a mock object for Room, specify the expected behavior and reserve the room for the given course.

Solution:

```
@RunWith(EasyMockRunner.class)
public class RoomReservationTest {

        @TestSubject
        private RoomReservation RoomReservation = new RoomReservation();

    1   @Mock
        private Room roomMock;

        @Test
        public void roomReservationSuccessfulTest() {

                Course course = new Course("EIST");
                String roomName = "Lecture Hall 1";

            2   expect(roomMock.getName()).andReturn(roomName);
            3   replay(roomMock);
            4   roomReservation.reserve(roomMock, course);

                String assignedCourseRoomName = course.getRoom().getName();
                assertEquals(assignedCourseRoomName, roomName);
        }
}
```

Correction Criteria:

- + 1 point for the Room mock attribute (1)
- + 1 point for the expect(…) statement (2)
- + 1 point for the replay(…) statement (3)
- + 1 point for calling reserve on roomReservation (4)

b) **[1p]** Which classes are part of the SUT? Justify your solution.

Solution: RoomReservation is the class which is unit tested, therefore it is the system under test (SUT).

Correction Criteria:
- + 0.5 points for the correct solution
- + 0.5 points for providing a valid justification

c) **[1p]** What are the collaborators? Justify your solution.

Solution: Course and Room are collaborating objects that participate in the unit test case. They are not tested themselves.

Correction Criteria:
+ 0.5 points for the correct solution
+ 0.5 points for providing a valid justification

d) **[1p]** Which classes are part of the test model? Justify your solution.

Solution: RoomMock is part of the test model because it mimics the behavior of the actual Room object just for testing purposes. RoomReservationTest is also part of the test model, because it is a test case.

Correction Criteria:
- + 0.5 points for the correct solution
- + 0.5 points for providing a valid justification