Exam Software Engineering Techniques

01-07-2024

- This exam is closed book. Hence, you cannot use any copies of slides, nor your own notes. Also you cannot use calculators or any other devices.
- This exam consists of 4 exercises, each consisting of several questions. Clearly indicate which question you answer.

1 Testing (10 points)

Suppose that you have a partial software implementation of the El Dorado game. You just finished your first implementation of the Hand cards of a Player, the Market Board, and that a Player can buy a card from the Market Board.

In Figure 1 some of the implementation's classes with the headers of some of its methods and constructors are shown. Above each method a short description of its functionality is given in comments.

```
class MarketBooard {
  // Constructor for initializing board with cards at start of game
  MarketBoard()

  // Iterate over all cards available for buying in the market
  Iterator<Card> availableCardsIterator()

  // Removes card from market
  removeCard(Card card)
}

class HandCards {
  // Constructor for initializing hand with random cards at start of game
  HandCards()

  // Constructor for initializing hand with t Traveler cards (each worth 1 gold)
  // and random other cards at start of game
  HandCards(int t)

  // Removes gold cards and returns true if amount of gold is in hand
  // returns false otherwise
  boolean spendGold(int amount)
}

class Card {
  //Creates a card with given name, power, color, and price
  Card(String name, int power, Color color, int Price)

  // Returns price of the card in gold
  int getPrice();
}

class Player {
  // Initializes player with hand
  Player(HandCards hand)

  // Returns the total amount of gold present in the hand
  int goldInHand()

  // Calls HandCards.spendGold MarketBoard.removeCard to buy card from market
  // Returns true if hand has enough gold and false otherwise
  boolean buy(Card card, MarketBoard market)
}
```

Figure 1: This figure shows some El Dorado classes with the headers of some of its methods and constructors. Above each method/constructor a short description of its functionality is given in comments.

Now you want to write an integration test for buying a card with hand cards from the market board. It should be checked that the gold spend from the hand of the Player is the same as the cost of the card from the market board.

(a) To implement the integration test, (3pts)

   (i) which classes should to be replaced by stubs? Shortly motivate your answer.

   (ii) which classes should be replaced by drivers? Shortly motivate your answer.

   (iii) which classes should be used in the integration test as-is, i.e. with their actual implementation? Shortly motivate your answer.

(b) Write the code for the integration test, using methods/constructors of Figure 1, and any methods/constructors for stubs or drivers that you need to introduce. If you include a line of code with a call to a stub or driver method/constructor, you need to include a comment line above that describes what that line of code achieves. (7pts)

2 Design patterns (12 points)

Suppose that you have implemented a `Board` class, consisting of the terrain tiles, terrain strips, end tile, and blockades. You now need to add the caves, such that a game can be player with caves on the board, but also without the caves. You want to use a design pattern to obtain the following flexibility: other classes should deal with the `Board` class in a uniform way, no matter whether it has caves or not. The current implementation, that always has caves, is displayed as a class diagram in Figure 2. Other classes deal with the `Board` class through its attributes and its method `getTerrainHexagon`.

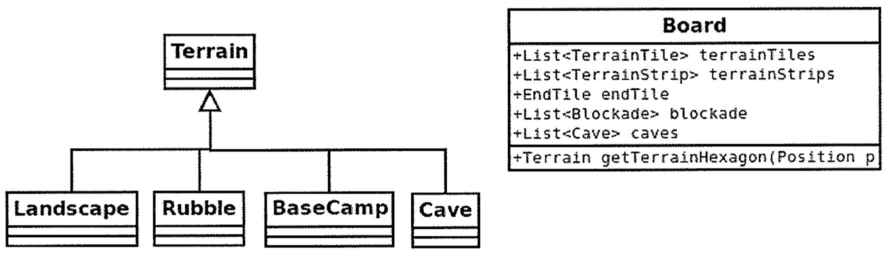
Terrain
Landscape
Rubble
BaseCamp
Cave
Board
+List<TerrainTile> terrainTiles
+List<TerrainStrip> terrainStrips
+EndTile endTile
+List<Blockade> blockade
+List<Cave> caves
+Terrain getTerrainHexagon(Position p

Figure 2: Class diagram of current implementation

(a) What design pattern would you apply to get the required flexibility for the `Board` class? Choose the most appropriate pattern out of the patterns from the lectures, i.e. ~~Observer pattern~~, ~~Composite pattern~~, ~~Strategy pattern~~, ~~Abstract Factory pattern~~, ~~Factory Method pattern~~, Decorator pattern, ~~Template Method pattern~~, Adapter pattern.
Motivate your choice! (2pts)

(b) Draw a class diagram for your implementation of the design pattern. Only include classes involved in the design pattern. (7pts)

(c) Motivate how the diagram of (b) implements your selected design pattern of (a) by drawing the class diagram of the general pattern, and stating which classes of the general diagram match with those of your diagram of (b). (3pts)

3 Software metrics and refactoring (10 points)

Suppose you implemented a class `GameController` corresponding to the class diagram of Figure 3.

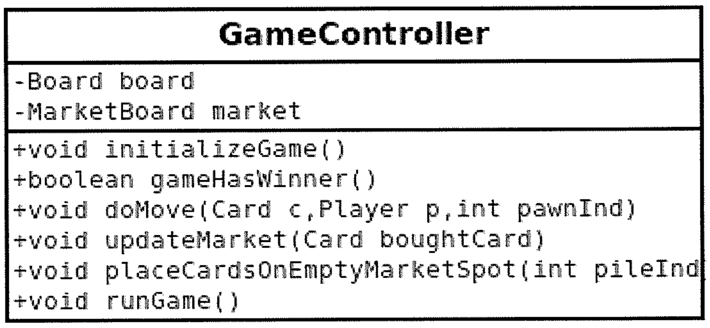
GameController
-Board board
-MarketBoard market
+void initializeGame()
+boolean gameHasWinner()
+void doMove(Card c,Player p,int pawnInd)
+void updateMarket(Card boughtCard)
+void placeCardsOnEmptyMarketSpot(int pileInd
+void runGame()

Figure 3: Class diagram of class `GameController`

(a) What maintainability guideline is violated? Shortly motivate your answer. (3pts)

(b) Explain how you would refactor this class. (6pts)

(c) Name the first refactoring step that you would apply towards achieving the refactoring you describe at (b). (1pt)

4 Debugging (10 points)

Figure 4 shows how the `spendGold` method of the `HandCards` class from Figure 1 was implemented.

```
 1  boolean spendGold(int amount) {
 2    List<Card> cardsToRemove = new ArrayList<>();
 3    while(amount > 0) {
 4      Card c = selectHighestGoldHandCard();
 5      if(c != null) { // gold in hand
 6        cardsToRemove.add(c);
 7        amount = amount - c.getPower();
 8      } else { // no gold in hand
 9        break;
10      }
11    }
12    if(amount <= 0) { //spend cards
13      removeCards(cardsToRemove);
14      return true;
15    } else { // not enough gold, keep cards
16      return false;
17    }
18  }
```

(a) What is the forward slice of the statement on line 2? Give the line numbers. (2pts)

(b) What is the backward slice of the statement on line 14? Give the line numbers. (2pts)

(c) After running some tests on the spendGold method, it turns out the method never returns `false` for any hand containing at least one gold card. Formulate the first two hypotheses for applying the scientific method to debug this issue. The first hypothesis should state the expected behaviour, before any test was run. The second hypothesis should give a clue on how to solve the bug, from the knowledge after running a test for the first hypothesis. Give the scientific method tables for both hypotheses. (6pts)