Test KEN1520 - Software Engineering RESIT 2023

Block introduction

Question order: Fixed

Question 1 − 2023_Resit_Architecture_OP − 137907.1.0

A hospital is planning to develop a new patient record management system to improve the efficiency and accuracy of patient care. The hospital authority has established strict confidentiality and privacy guidelines for the system, which must use advanced encryption and access control mechanisms to protect patient data with minimal human intervention.
As the project manager for the software development process, which of the following process models would you choose for this project, considering the need for adaptability and flexibility in the face of evolving requirements and technologies?

Explain your choice in 5 to 8 sentences, give both a benefit and a possible downside for your choice of approach.

a. The waterfall method: a sequential, linear approach where each phase of the project must be completed before moving to the next one

b. The iterative incremental model: a cyclic process that involves developing the software in small increments, with regular feedback and adjustments

c. An agile method such as Scrum: an iterative, adaptive approach that focuses on collaboration, self-organization, and continuous improvement

Grading instruction

Correct explanation (Number of points: 10)

Question 2 – 2023_Resit_areEqual_Test_MC – 137921.1.1

Consider the following test case:

```
@Test public <T> void areEqual(T[] container_a, T[] container_b) throws DifferentElementException {
  boolean condition = container_a.length == container_b.length;
  if (condition) {
    for (int i = 0; i < container_a.length; ++i) {
      if (container_a[i] != container_b[i]) {
        throw new DifferentElementException("Element " + i + " differs.");
      }
    }
  }
  assertEquals(true, condition);
}
```

Which of the following statements are true?

A	It tests whether two generic containers contain the same number of elements.
B	It tests whether two generic containers contain exactly the same elements.
C	When thrown, the exception DifferentElementException clearly shows all failing indices.
D	It can be reused to test a variety of containers.

Question 3 – 2023_Resit_BinaryS_Spec_OP – 137932.1.0

Consider the following specification:

```
/**
 * Find a value in an array
 * @param arr sorted array to search
 * @param key value to search for
 * @return index of the search key, if it is contained in the array; otherwise -1
 */
static int binarySearch(int[] arr, int key) {
  int l = 0;
  int r = arr.length();
  while (l < r) {
    int m = (l + r) / 2;
    if (key > arr[m]) {
      l = m + 1;
    }
    else {
      r = m;
    }
  }
  return l;
}
```

Grading instruction

Criterion 1 (Number of points: 3)

Give an input to demonstrate that binarySearch does not satisfy the specification.

Criterion 2 (Number of points: 3)

Change the specification so that the binarySearch function satisfies it.

Question 4 − 2023_Resit_Bob_Alice_Git_OP − 137922.1.2

Bob, an innocent maintainer, accidentally pushed a very large file to the repository he shares with Alice. At present, Alice is unaware of this and is working on her local repository.

Answer the following **three** questions:

1. What steps should Bob take to rectify the situation if he had pushed the big file to a branch other than the master branch?
2. Unfortunately, Bob was unaware and merged it with the master branch. Later, Alice fetches the repository. Since Alice fetched while not being on any of the affected branches, will her computer download the big file?
3. Bob now realizes the gravity of the situation. How can he resolve this worsened scenario?

Grading instruction

Criterion 1 (Number of points: 3)

Criterion 2 (Number of points: 3)

Criterion 3 (Number of points: 3)

Question 5 – 2023_Resit_CodeSmell_OP – 137911.1.1

Below is a code snippet representing a basic library management system. Analyze the code and identify one code smell present in the code. Suggest a refactoring technique to improve the design of the code, and provide a brief explanation of how the refactoring technique improves the code. Although there may be more than one smell in the code, you **only must choose and solve one**. You may use source code in your answer to illustrate your solution.

```
class LibraryMember {
    private String name;
    private List<Book> borrowedBooks;

    public LibraryMember(String name) {
        this.name = name;
         this.borrowedBooks = new ArrayList<>();
    }

    public void borrowBook(Book book) {
         borrowedBooks.add(book);
    }

    public String generateSummary() {
        int fictionBooks = 0;
        int nonFictionBooks = 0;
         StringBuilder result = new StringBuilder("Summary for " + name + "\n");

        for (Book book : borrowedBooks) {
            switch (book.getCategory()) {
                 case Book.FICTION:
                     fictionBooks++;
                     break;
                 case Book.NON_FICTION:
                     nonFictionBooks++;
                     break;
            }
             result.append("\t").append(book.getTitle()).append("\t").append(book.getCategory()).append("\n");
        }

         result.append("Total Fiction books: ").append(fictionBooks).append("\n");
         result.append("Total Non-Fiction books: ").append(nonFictionBooks).append("\n");
        return result.toString();
    }
}
```

Which of the following code smell can you identify in the code snippet? Choose **one**.

- **Long Method:** Excessively lengthy, complex functions.
- **Large Class:** Overstuffed class with many responsibilities.
- **Data Clumps:** Frequently occurring grouped variables.
- **Primitive Obsession:** Overusing primitive data types.
- **Divergent Change:** Class modified for multiple reasons.
- **Shotgun Surgery:** Small changes spread across classes.

Which of the following refactoring techniques would be most appropriate to address the identified code smell? Choose **one.**

- Extract Method
- Extract Class
- Introduce Parameter Object
- Replace Primitive with Object
- Extract Subclass
- Move Method
- Replace Conditional with Polymorphism

Grading instruction

Correctness (Number of points: 10)

Question 6 – 2023_Resit_Payment_MC – 137908.2.0

You are developing a payment processing system for an e-commerce website. The system currently processes payments via credit card, but you anticipate that in the future, it might need to support other payment types, such as PayPal or Bitcoin. Based on the SOLID principles, which of the following design approaches would best ensure that the system is easy to extend and maintain?

A Create an interface or abstract base class for the payment processing system, implement separate classes for each payment type, and ensure that higher-level modules depend on the interface or abstract base class rather than specific payment processing classes.

B Implement a base class for the payment processing system with a concrete `processPayment()` method and inherit from this class for each new payment type, overriding the method as necessary.

C Create a single class with multiple methods, one for each payment type (credit card, PayPal, Bitcoin), and modify the class whenever a new payment type is needed.

D Use a procedural approach, writing separate functions for each payment type, and call the appropriate function based on the payment type needed.

E Implement a single class with a switch statement or conditional logic that determines which payment type to use based on an input parameter or configuration setting.

Question 7 – 2023_Resit_replace_debug_RANK – 137945.2.0

```java
/**
* Replace ocurrences of key with val in the list arr.
* @param arr list of integers
* @param key element to be replaced
* @param val value to replace key with
* @return a list where every ocurrence of key is replaced with val.
*/
public static List<int> replace(List<int> arr, int key, int val) {
  ...
}
```

A user files a bug reporting that invoking replace([0, 1, 2, 2], 2, 4) results in [0, 1, 2, 4]. Set the right order for the following debugging actions:

1 Reproduce the bug.

2 Use a debugger.

3 Watch memory locations.

4 Change the code from using an List<int> to using int[].

Question 8 − 2023_Resit_Shopping_Coupling_MR − 137915.1.0

Consider the following Java classes in different packages of a software system:

```
package com.example.cart;
public class ShoppingCart {
  private List<CartItem> cartItems;

  private ShippingService shippingService;
  // ... other fields and methods
}

package com.example.cart;
public class CartItem {

  private Item item;
  // ... other fields and methods
}

package com.example.inventory;
public class Item {
  private List<Rating> ratings;

  private ItemType itemType;
  // ... other fields and methods
}

package com.example.inventory;
public class ItemType {
  private List<Item> items;
  // ... other fields and methods
}

package com.example.delivery;
public class ShippingService {
  private Carrier carrier;
  // ... other fields and methods
}

package com.example.delivery;
  public class Carrier {
  // ... other fields and methods
}
```

Which of the following two changes to the codebase promote loose coupling and high cohesion?

A Moving the ItemType and Item classes into the com.example.cart package

B Moving the ShoppingCart and CartItem classes into the com.example.delivery package

C Creating a Logistics package and moving the ShippingService and Carrier classes into it

D Extracting an interface from the ShippingService class and passing an instance of the interface to the ShoppingCart class constructor

E Removing the ratings field from the Item class and creating a separate Rating class in a com.example.rating package

F Changing the ShippingService class to be a singleton and using a static factory method to create its instance

Question 9 – 2023_Resit_Social_Media_FILL – 137910.2.0

A colleague wrote the following class for a social media platform:

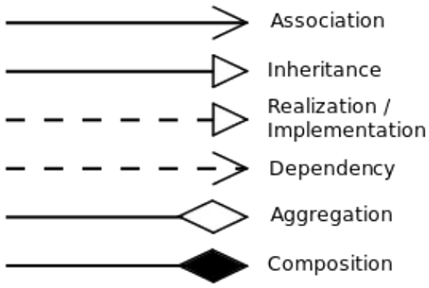
```
public class User {
  private String username;
  private String password;
  private String email;
  private List<String> friends;
  public User(String username, String password, String email) {
      this.username = username;
      this.password = password;
      this.email = email;
      this.friends = new ArrayList<>();
  }

  public void addFriend(User friend) {
       this.friends.add(friend.getUsername());
       friend.friends.add(this.getUsername());
  }

  public List<String> getFriends() {
      return this.friends;
  }

  public String getUsername() {
      return this.username;
  }

  public String getEmail() {
      return this.email;
  }

  // getters and setters for password
}
```

This class violates the single responsibility principle, which states that a class should have only one reason to change. To adhere to this principle, you could create separate classes for authentication and authentication , with only the relevant methods for each class.

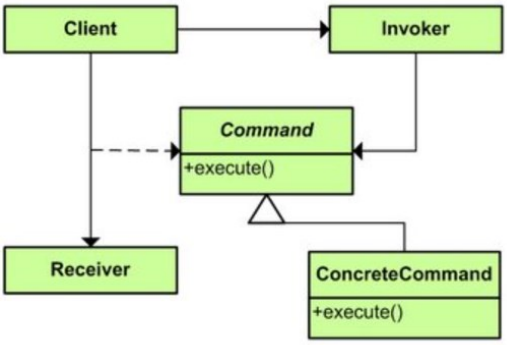
[Alphanumeric]	[Alphanumeric]	[Alphanumeric]
single responsibility	**authentication**	**authentication**
responsibility	authorization	authorization
single	login	login
first	profile	profile
	friends list	friends list
	friends	friends
	followers	followers

Design Patterns

Question order: Fixed
Cheat Sheet:



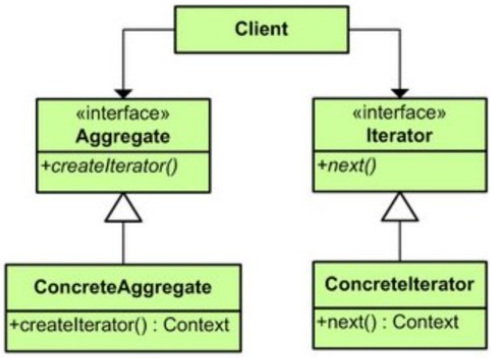
Design Patterns:



Command

Type: Behavioral

What it is:
Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.



Iterator

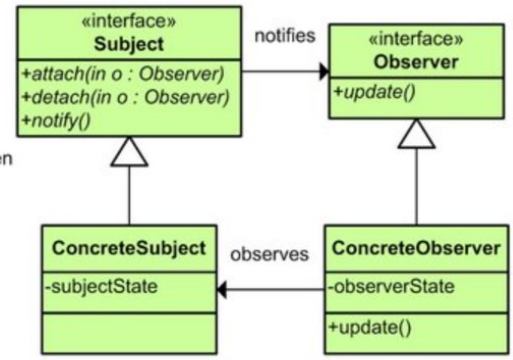
Type: Behavioral

What it is:
Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.

Observer

Type: Behavioral

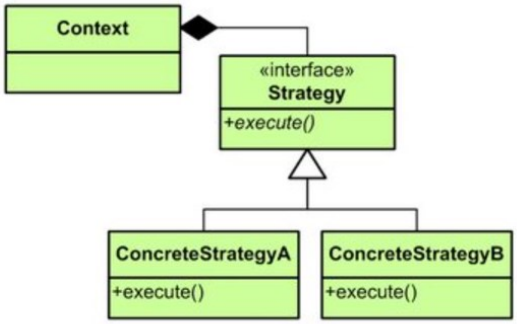
What it is:
Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.



Strategy

Type: Behavioral

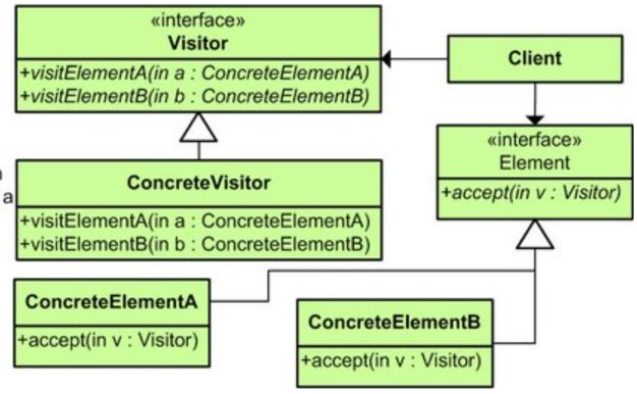
What it is:
Define a family of algorithms, encapsulate each one, and make them interchangeable. Lets the algorithm vary independently from clients that use it.



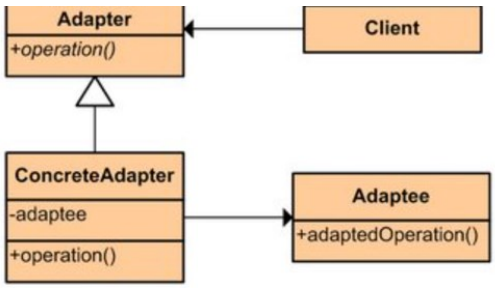
Visitor

Type: Behavioral

What it is:
Represent an operation to be performed on the elements of an object structure. Lets you define a new operation without changing the classes of the elements on which it operates.

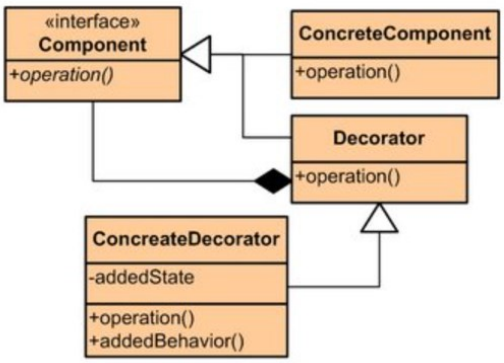


Adapter



Type: Structural

What it is:
Convert the interface of a class into another interface clients expect. Lets classes work together that couldn't otherwise because of incompatible interfaces.



Decorator

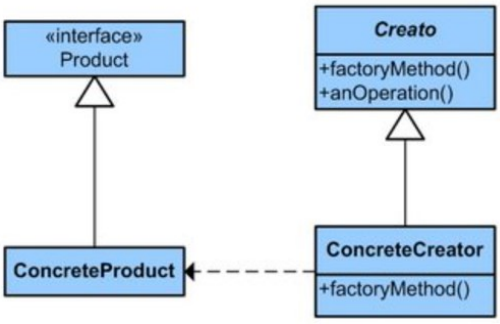
Type: Structural

What it is:
Attach additional responsibilities to an object dynamically. Provide a flexible alternative to sub-classing for extending functionality.

Factory Method

Type: Creational

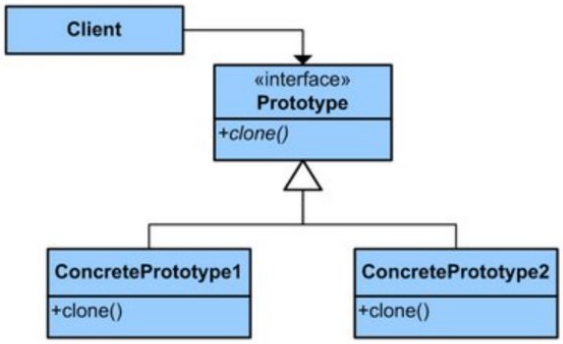
What it is:
Define an interface for creating an object, but let subclasses decide which class to instantiate. Lets a class defer instantiation to subclasses.



Prototype

Type: Creational

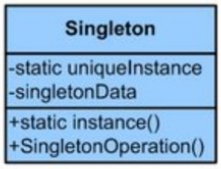
What it is:
Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.



Singleton

Type: Creational

What it is:
Ensure a class only has one instance and provide a global point of access to it.



Question 10 − 2023_Resit_DP_Match − 137912.2.0

For each of the following scenarios, select the design pattern that would be most appropriate to apply:

Adapter	You are building a new feature for a mobile application that needs to access and display data from different sources such as a database or an external API. However, the application code should not be tightly coupled to any specific data source.
Strategy	You are working on a program that needs to execute different algorithms at runtime, depending on user input. The algorithms can be complex and may need to be changed frequently.
Prototype	You are developing a game that allows players to customize their character by selecting different clothing items, hairstyles, and accessories. You want to be able to add new customization options without having to modify the existing codebase.
Factory	You are creating a system to manage different types of vehicles (e.g., cars, trucks, motorcycles). Each vehicle has its own set of properties (e.g., fuel type, engine size, number of doors), but there are also some common properties (e.g., make, model). You want to be able to add new types of vehicles without having to modify the existing codebase.

Question 11 − 2023_Resit_Graphic_Editor_DP_MC − 137913.1.0

Consider a graphics editor application that allows users to manipulate various graphical objects, such as circles, rectangles, and lines. The application needs to have the ability to export these graphical objects to different file formats, such as SVG, PNG, and PDF, as well as display the total area of all the objects in the scene. The application should be designed to allow easy addition of new graphical objects and export formats without modifying existing code.

Which of the following design patterns is most suitable for implementing the behavior described above?

A Singleton

B Observer

C Strategy

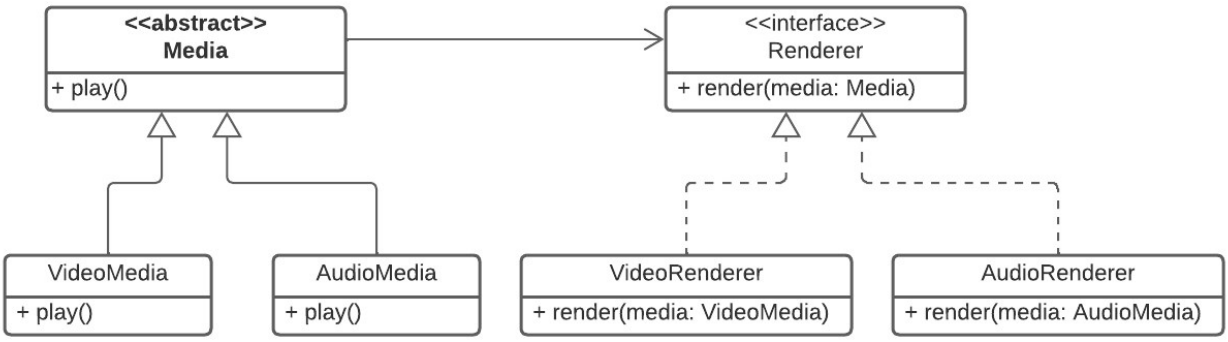
D Visitor

E Command

Question 12 – 2023_Resit_Media_DP_MC – 137920.1.0

Consider a media player application that allows users to play various types of media files (audio, video, etc.). The media player's key abstraction is the media file, which can be played and controlled through a set of operations.

The interface for media files is defined by an abstract class called Media. The application defines a subclass of **Media** for each type of media file: an **AudioMedia** class for audio files, a **VideoMedia** class for video files, and so forth.



This class diagram above illustrates the use of a design pattern. It shows how Play requests, declared in class Media, are converted to Render requests defined in the **Renderer** interface. Since **VideoRenderer** and **AudioRenderer** implement **Renderer**, the media player renders different types of media based on their type.

Which design pattern is represented in this class diagram?

- **A** Decorator
- **B** Factory
- **C** Command
- **D** Singleton
- **E** Strategy

Error Types

Question order: Fixed
Some of the following pieces of code contain errors. Determine whether they will result in static errors, dynamic errors, or whether the program will run with no errors.

Question 13 – 2023_Resit_Error_Type_1_MC – 137923.1.0

```
int x = 5;
String s = x + "a";
```

- **A** Static Error
- **B** Dynamic Error
- **C** No Errors

Question 14 − 2023_Resit_Error_Type_2_MC − 137924.2.0

```
int a = 10;
int b = 0;
double c = a % b;
```

- **A** Static Error
- **B** Dynamic Error
- **C** No Errors

Question 15 − 2023_Resit_Error_Type_3_MC − 137928.2.0

```
int b = 0;
for (double a = 10; a > 1; a /= 100) {
  b++;
}
```

- **A** Static Error
- **B** Dynamic Error
- **C** No Errors

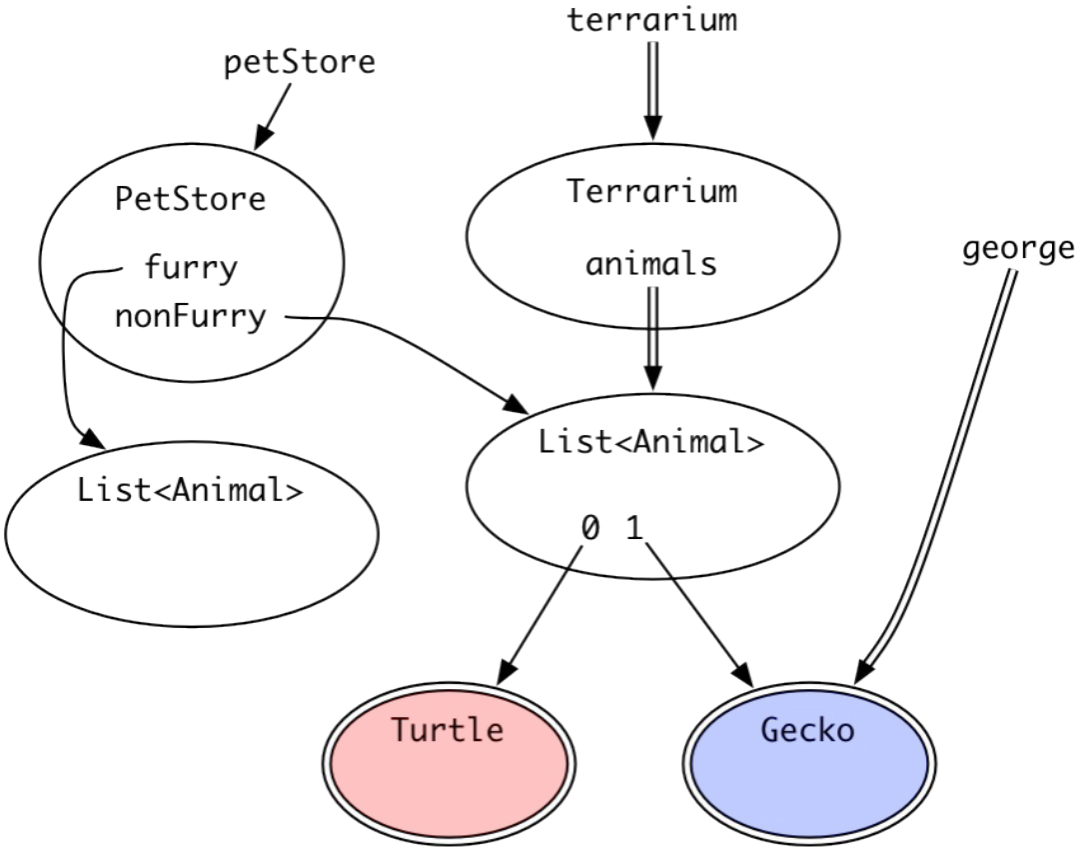
Question 16 − 2023_Resit_Error_Type_4_MC − 137929.2.0

```
int a = 2;
boolean b = a;
```

- **A** Static Error
- **B** Dynamic Error
- **C** No Errors

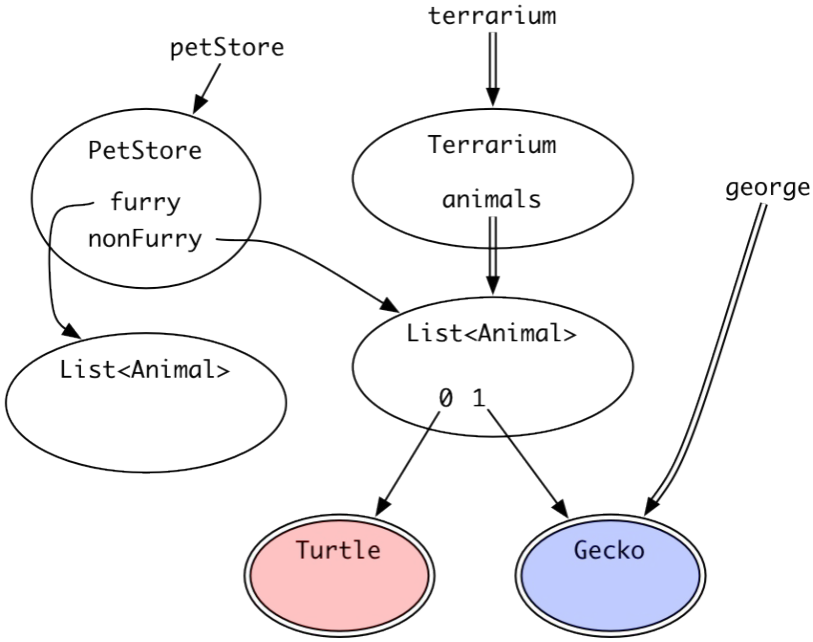
Snapshot Diagram

Question order: Fixed
The following two questions refer to the snapshot diagram below:



Question 17 − 2023_Resit_petStore_MC − 137937.1.0

Consider the following snapshot diagram, taken from the labs:



Can the variable terrarium affect how petStore accesses the Turtle?

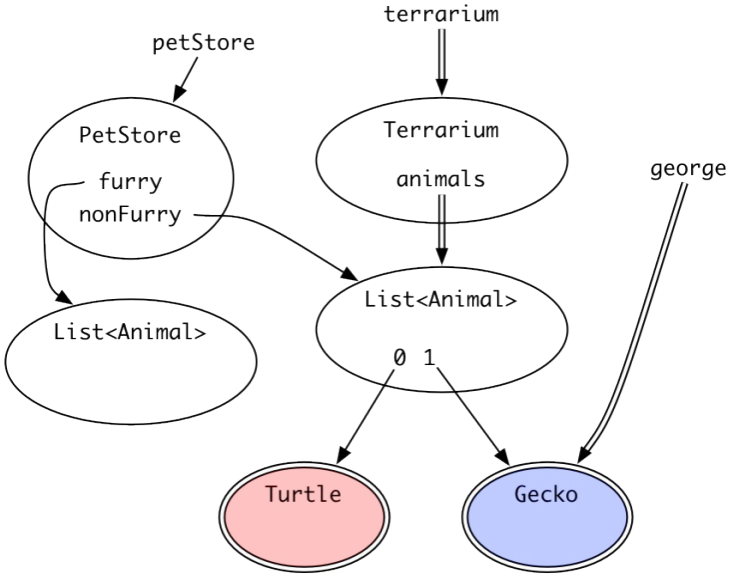
A No, because the Turtle object is immutable.

B No, because the animals reference is immutable.

C Yes, because the List object is mutable.

D Yes, because the reference from list index 0 to Turtle is mutable.

Question 18 − 2023_Resit_PetStore_Ref_MC − 137944.1.3



Regarding the Pet Shop snapshot diagram, which of the following statements regarding references are true?

Your score for this question is based on the proportion of correct answers, while considering the proportion of incorrect answers.

A furry and nonFurry are references that can be updated to point to other objects.

B The object terrarium cannot modify the variable animals because it is an immutable reference.

C In Java, an immutable object can be achieved by declaring it final.

D In Java, an immutable reference requires that the variable be declared final and that its API does not expose any setters that could alter the object.