Name: David A. DiPesa

Course: CS-230-11604-M01: Operating Platforms

### Instructor: Alexander Dubinski, MS CS, MBA

Chapter 1 Questions

Question 1: What are the two basic types of UML diagrams?

The two basic types of UML diagrams are behavioral and structural.

Question 2: Why was UML initially created?

UML was developed in 1994 to document object-oriented systems. Since that time, UML has become a standard tool for software engineers, software developers and businesses. UML is used to design software and, after programming, also provides a visual record for the system which can be included as part of formal software documentation.

Question 3: List four behavioral UML diagrams:

Four behavioral UML Diagrams are:

Activity diagrams, Interaction diagrams, State machine diagrams and Use case diagrams

Question 4: Which UML diagram provides a system's process flow?

The diagram we’re looking for is called an Activity Diagram. Activity diagrams illustrate the **flow of processes** in a system. This type of diagram is used to visually document activities within a system, also referred to as a system's procedures or dynamic components.

Question 5: Which UML diagram documents the interactions between a system and its users?

In UML, use-case diagrams model the behavior of a system and help to capture the requirements of the system. Use-case diagrams describe the high-level functions and scope of a system. These diagrams also identify the interactions between the system and its actors.

Question 6: What is a UML actor?

Actor: The stick figure is referred to as an actor. In our example, student and instructor were both actors. These are the users that use your system. Often, there are multiple user roles in a system.

Also, found online: Actors are **the users that interact with a system**. An actor can be a person, an organization, or an outside system that interacts with your application or system. They must be external objects that produce or consume data.

Question 7: What type of UML diagram illustrates system components?

Component diagrams provide a visual representation of a system's physical components. The following example illustrates the three physical components of the system:

An inventory database, a customer database and an order component

Question 8: What is the most commonly used UML diagram?

The class diagram is the most commonly used UML diagram, as it provides a visual description of a system's objects. Consider that, in Java, everything is an object, so you can see the relevance and reason as to why this particular diagram is so widely used. Class diagrams do more than just display objects – they visually depict their construction and relationships with other classes.

Question 9: Which UML diagram shows the runtime structure of a system?

The Composite structure UML diagram shows the runtime structure of a system. This diagram can be used to show the internal components of a class

Question 10: Which UML diagram visually documents a system's hardware and software?

Deployment diagrams provide a visual representation of a system's hardware and software. Physical hardware components are illustrated, along with the particular software components that are on them. The hardware components are represented as nodes, and software is represented as an execution environment.

Summary of Chapter 1

The Universal Modeling Language (UML) is used to create visual documentation of our systems. This can be used to design a system as well as document a system. UML is widely used by software engineers, software developers, and other professionals.

Two of the 14 UML diagrams are behavioral and structural. Behavioral diagrams illustrate how system components interact to form a system and include activity diagrams, interaction diagrams, state machine diagrams, and use case diagrams. There are several types of interaction UML diagrams, including sequence diagrams, communication diagrams, and timing diagrams.

Structural diagrams illustrate components of a system and include class diagrams, component diagrams, composite structure diagrams, deployment diagrams, object diagrams, and package diagrams.

In the next chapter, Object-Oriented Design Patterns, we will explore intermediate and advanced concepts and approaches to object-oriented programming and their applicability to design patterns. A review of the fundamental concepts of object-oriented programming will help to ensure a deep conceptual understanding of object-oriented programming. An overview of object-oriented programming-related design pattern principles will also be provided.

Summary of Chapter 2

This chapter started with an introduction to OOP. Portability, inheritance, encapsulation, and polymorphism were deemed to be the primary benefits of OOP and were explored. A sample OOP class was examined in detail. That class, the Bicycle class, was used to demonstrate key OOP concepts to include instance variables, the this reference, accessors, mutators, driver class, constructors, overloading, and method call chaining. The chapter ended with a look at key OOP principles – create concise objects, encapsulate to protect, and purposeful inheritance.

In the next chapter, Behavioral Design Patterns, we will explore the behavioral design pattern category and its individual design patterns of chain of responsibility, mediator, memento, null object, observer, state, strategy, template method, and visitor. We will examine the programming challenges and design patterns that resolve them.

Question 1: What are the primary benefits of OOP languages?

There are many benefits of OOP. Chief among them are the following: Portability, Inheritance, Encapsulation and Polymorphism

Question 2: Which OOP construct lends itself well to portability?

The class structure lends itself well to portability.

Question 3: What refers to the hidden nature of an object's data components?

Encapsulation refers to the hidden nature of an object's data components. We know that the data is there, but cannot access it directly because external entities cannot directly interact with that data.

Question 4: What is the definition of polymorphism in relation to OOP?

In OOP, polymorphism states that different types of objects can be accessed via a common interface. This can be achieved by writing overloaded constructors and methods and is covered later in this chapter. Polymorphism can also be achieved by having subclasses override superclass methods.

Question 5: What is the “this” reference used for?

In Java, we can use the “this” keyword as a reference to the current object. For example, the following code snippet sets the color of the current Bicycle object:

public void setColor(String theColor) {

this.color = theColor;

}

The preceding setColor method accepts a String parameter and assigns it to the color instance variable of the current object.

Question 6: What is an accessor method?

Accessor methods are those that allow an object's data to be accessed. These methods can get the data, but not change it. This is a great way to protect the data from being changed. Accessor methods are also referred to as Getters methods.

Question 7: What is a mutator method?

Mutator methods, also known as setter methods, allow the object's instance variables to be changed.

Question 8: What is a constructor?

Constructors are a special kind of method that are run when an object is initialized. We use constructors to set up an object with default values.

Question 9: How are overloaded constructors or methods unique?

Each overloaded constructor is used to perform different task in the class. The Java compiler identifies the overloaded constructors **on the basis of their parameter lists, parameter types and the number of input parameters**. Hence, the constructors that are overloaded should have different signatures.

Question 10: In what direction are chained method calls executed?

OOP affords us the opportunity to use method call chaining. That is the process of making multiple method calls in a single statement. Here is the syntax:

object.method1().method2().method3().method4();

Using the syntax provided previously, we can walk through the process of method call chaining. The object first calls method1(), which returns the calling object. Next, that returned object calls method2(), which returns the calling object. You can see the process here. It works from left to right:

