**M S RAMAIAH INSTITUTE OF TECHNOLOGY**

(Autonomous Institute, affiliated to VTU)

**DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING**

|  |  |  |  |
| --- | --- | --- | --- |
| **Course:** | OO Analysis and Design Patterns | **Semester:** | 6 A, B, C |
| **CIE:** | Test – II | **Max Marks:** | 30 |
| **Date:** |  | **Course Code:** | IS 62B |

**Portions for Test:**  Adapter Pattern, Strategy Pattern, Abstract Factory Pattern, Principles and Strategies of Design Patterns, Commonality and Variability Analysis:

**Instructions to Candidates:** Each question in Part A carries 2 marks and Part B carries 1

Mark.

|  |  |  |
| --- | --- | --- |
| **Part A** | | |
| **Sl. #** | **Question** | **COs** |
| **1** | What are the types of adapter pattern are ------ and -------- | **CO3** |
| a)Static , Dynamic |
| b)object, Class |
| c)private ,Public |
| d)volatile ,Nonvolatile |
| **Answer: b)** |
| **2** | Traditional view of objects deals with -----------and -------- | **CO3** |
| a) Data ,Methods |
| b) Data, Objects |
| c) implementation of methods, Access Specifies |
| d) Designing the objects, Object type |
| **Answer: a)** |
| **3** | Modern view of objects focuses on --------- with –------ | **CO3** |
| a) Things with responsibilities |
| b) Things with data |
| c) Things with pointers |
| d) Access management |
| **Answer: a)** |
| **4** | Approaches of strategy pattern are -------- and ------------ | **CO3** |
| a)Switches and Inheritance |
| b)Nested loops and Abstraction |
| c)If loops and Polymorphism |
| d)While and accessibility |
| **Answer: a)** |
| **5** | Inheritance hierarchies resulted from specialization has the following drawbacks.  They are -------- and -------- | **CO3** |
| a)Redundancy, Hard to test |
| b)High cohesion ,Low coupling |
| c)High coupling, Low cohesion |
| d) High cohesion, High coupling |
| **Answer: a)** |
|  |
| **6** | Match the key features of Abstract Factory Pattern   |  |  | | --- | --- | | 1. Problem | 1. Coordinates the creation of families of objects | | 1. Solution | 1. Define abstract class and then implement the concrete class for each family | | 1. Consequence | 1. Families of related objects need to be instantiated | | 1. Implementation | 1. Isolates the rules of which objects to use from logic of how to use these objects | | **CO3** |
|  |
| 1. A – 1, B – 3, C – 2, D - 4 |
| 1. A – 3, B – 1, C – 4, D – 2 |
| 1. A – 2, B – 4, C – 1, D – 3 |
| 1. A – 4, B – 2, C – 3, D – 1 |
| **Answer: b)** |
|  |  |  |
| **7** | Which of the following statements on “dependency – inversion principle” are correct?  Statement 1: High level modules should not depend on low-level modules  Statement 2: Both high-level and low-level modules should depend upon abstractions  Statement 3: Abstractions depend on details  Statement 4: Details should depend upon abstractions | **CO4** |
| 1. Statements 1 and 3 only are correct |
| 1. Statements 1 and 2 only are correct |
| 1. All the statements except Statement 4 are correct. |
| 1. All the statements except Statement 3 are correct. |
| **Answer: d)** |
|  |  |  |
| **8** | Match the key features of Bridge Pattern   |  |  | | --- | --- | | 1. Problem | 1. Encapsulation in abstract class | | 1. Solution | 1. Define an interface for all implementations to use and have the derivations of the abstract class use that | | 1. Consequence | 1. Decoupling of implementations from the objects that use them increases extensibility | | 1. Implementation | 1. Derivations of an abstract class must use multiple implementations without causing an explosion in the number of classes | | **CO3** |
| 1. A – 1, B – 3, C – 2, D - 4 |
| 1. A – 3, B – 1, C – 4, D – 2 |
| 1. A – 2, B – 4, C – 1, D – 3 |
| 1. A – 4, B – 2, C – 3, D – 1 |
| **Answer: d)** |
|  |  |  |
| **9** | It is well-known that Bridge Pattern has lot many object oriented principles. Choose the proper reason for the principle “One rule – one place.”  Reason 1: The abstract class often has the methods that actually use the implementation objects. The derivations of the abstract class call these methods. This allows for easier modification if needed, and allows for a good starting point even before implementing the entire pattern.  Reason 2: In case there are many shapes, each shape object took the task of drawing its own shape.  Reason 3: All classes must be inherited from the abstract class which binds all those concepts together. | **CO3** |
| 1. Reasons 1 and 2 |
| 1. Reason 1 only |
| 1. Reason 2 only |
| 1. Reason 3 only |
| **Answer: b)** |
|  |  |  |
| **10** | Consider the following class diagram    Which of the following Java Code fragments best represent this class diagram? | **CO3** |
|  | a)  abstract class ResFactory  {  abstract public DisplayDriver getDispDrvr();  abstract public PrintDriver getPrtDrvr();  }  class LowResFact extends ResFactory  {  public DisplayDriver getDispDrvr()  {  return new LRDD();  }  public PrintDriver getPrtDrvr()  {  return new LRPD();  }  }  class HighResFact extends ResFactory  {  public DisplayDriver getDispDrvr()  {  return new HRDD();  }  public PrintDriver getPrtDrvr()  {  return new HRPD();  }  } |  |
|  | b)  abstract class ResFactory  {  abstract public DisplayDriver getDispDrvr();  abstract public PrintDriver getPrtDrvr();  }  class LowResFact extends ResFactory  {  public DisplayDriver getDispDrvr()  {  return new LRPD();  }  public PrintDriver getPrtDrvr()  {  return new LRDD();  }  }  class HighResFact extends ResFactory  {  public DisplayDriver getDispDrvr()  {  return new HRDD();  }  public PrintDriver getPrtDrvr()  {  return new HRPD();  }  } |  |
|  | c)  abstract class ResFactory  {  abstract public DisplayDriver getDispDrvr();  abstract public PrintDriver getPrtDrvr();  }  class LowResFact extends ResFactory  {  public DisplayDriver getDispDrvr()  {  return new LRPD();  }  public PrintDriver getPrtDrvr()  {  return new LRDD();  }  }  class HighResFact extends ResFactory  {  public DisplayDriver getDispDrvr()  {  return new HRPD();  }  public PrintDriver getPrtDrvr()  {  return new HRDD();  }  } |  |
|  | d)  abstract class ResFactory  {  abstract public DisplayDriver getDispDrvr();  abstract public PrintDriver getPrtDrvr();  }  class LowResFact extends ResFactory  {  public DisplayDriver getDispDrvr()  {  return new LRDD();  }  public PrintDriver getPrtDrvr()  {  return new LRPD();  }  }  class HighResFact extends ResFactory  {  public DisplayDriver getDispDrvr()  {  return new HRPD();  }  public PrintDriver getPrtDrvr()  {  return new HRDD();  }  } |  |
|  | **Answer: a)** |  |

|  |  |  |
| --- | --- | --- |
|  | **Part B** |  |
| **Sl. #** | **Question** | **COs** |
| 1 | Which of the below is not a valid design pattern? | **CO2** |
| a)Facade |
| b)Adapter |
| c)Strategy |
| d) Java |
| **Answer: d)** |
|  |  |  |
| 2 | The recurring aspects of designs are called design | **CO2** |
| a)Patterns |
| b)Documents |
| c)Structures |
| d)Methods |
| **Answer: a)** |
|  |  |  |
| 3 | Which of the following describes the Adapter pattern correctly? | **CO2** |
| a)[This pattern builds a complex object using simple objects and using a step by step approach](javascript:void(0);) |
| b) This pattern refers to creating duplicate object while keeping performance in mind. |
| c) This pattern works as a bridge between two incompatible interfaces. |
| d)[This pattern is used when we need to decouple an abstraction from its implementation](javascript:void(0);) |
| **Answer: c)** |
|  |  |  |
| 4 | Which of the following class acts as a wrapper with the desired interface? | **CO2** |
| a) Strategy |
| b) Adapter |
| c)Facade |
| d)Singleton |
| **Answer: b)** |
|  |  |  |
| 5 | A system has right data and behavior but the wrong interface. Which pattern is appropriate here? | **CO2** |
| a) Adapter |
| b)Template |
| c)Abstract Factory |
| d)Object pool |
| **Answer: a)** |
|  |  |  |
| 6 | Are there any preexisting classes in Adapter pattern? | **CO2** |
| a)Yes |
| b)No |
| **Answer: a)** |
|  |  |  |
| 7 | What are the perceptive of commonality and variability class? | **C03** |
| a)Conceptual |
| b)Specification |
| c)Implementation |
| d)all of the above |
| **Answer: d)** |
|  |  |  |
| 8 | The pattern that enables to use different business rules or algorithms depending on the context is | **C03** |
| a)Facade pattern |
| b)Strategy pattern |
| c)Template |
| d)Simple Factory |
| **Answer: b)** |
|  |  |  |
| 9 | In an international e-commerce system there are different tax algorithms for different countries which pattern would be most appropriate for this case study? | **C03** |
| a) Strategy pattern |
| b)Abstract Factory |
| c) command |
| d)Bridge |
| **Answer: a)** |
|  |  |  |
| 10 | In the protypical strategy pattern the responsibility for selecting the particular implementation is done by ------ object | **C03** |
| a) Server |
| b)Context |
| c)Concrete |
| d) Client |
| **Answer: d)** |

**More sample questions**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Question and Answer** | **CO** |
| **1** | The intent of \_\_\_\_\_\_ pattern is to \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | **CO3** |
| 1. Strategy, Decouple abstraction from its implementation |
| 1. Bridge, Define family of algorithms, encapsulate each one and make them interchangeable |
| 1. Bridge, Decouple abstraction from its implementation |
| 1. Strategy, Define family of algorithms, encapsulate each one and make them interchangeable |
| **Answer: c)** |
|  |  |  |
| **2** | Pick the correct statement | **CO3** |
|  | 1. Bridge pattern often incorporates strategy pattern |
|  | 1. When two or more patterns are integrated, the result is complex design pattern |
|  | 1. When designing with Bridge pattern, the implementation’s interface should be designed considering the different derivations of the abstract class it will have to support |
|  | 1. Decoupling of implementation from objects that use them decreases extensibility |
|  | **Answer: c)** |
|  |  |  |
| **3** | In Abstract Factory Pattern, what does Abstract Factory participant do? | **CO3** |
|  | 1. Isolates rules of which objects to use from the logic of how to use these objects |
|  | 1. Defines interface for how to create each member of the family of objects required |
|  | 1. Coordinates creation of families of objects |
|  | 1. Integrates different design patterns |
|  | **Answer: b)** |
|  |  |  |
| **4** | The Open-Closed Principle can be stated as: | **CO4** |
|  | 1. Modules, methods and classes must be closed for extension and open for modification |
|  | 1. Modules, methods and classes must be open for extension and closed for modification |
|  | 1. Modules, methods and classes must be open for extension and open for modification |
|  | 1. Modules, methods and classes must be closed for extension and closed for modification |
|  | **Answer: b)** |
|  |  |  |
| **5** | “The patterns are useful as far as they go but use them only as a help to your thinking and not as a prescription for a solution.”  The given statement states \_\_\_\_\_\_\_\_\_\_\_\_\_\_. | **CO4** |
|  | 1. Principle of Encapsulating Variation |
|  | 1. Principle of Designing from Context |
|  | 1. Principle of Healthy Skepticism |
|  | 1. Open-Closed Principle |
|  | **Answer: c)** |