TT TEXTBOOK TESTBANKS

Test Bank for Head First Design Patterns 2nd Freeman

TEST BANK SAMPLE PREVIEW

PUBLISHER

OReilly

COPYRIGHT

2020

RESOURCE TYPE

Test Bank

ISBN

9781492078005

Test Bank for Head First Design Patterns 2nd Edition by Freeman, Robson

ISBN: 9781492078005

Head First Design Patterns, 2nd Edition

ISBN 9781492078005 | Chapter 1: Intro to Design Patterns: Welcome to Design Patterns

Total Questions: 100

Multiple Choice

Q1.
In the chapter, what is the primary benefit of using design patterns instead of relying only on code reuse?

A) Experience reuse ✓ Correct
B) Compiler optimization
C) Database normalization
D) Automatic testing

Rationale: The chapter states that with patterns you get experience reuse rather than just code reuse.

Q2.
What was the original structural design of the SimUDuck application?

A) A set of unrelated duck classes with no common parent
B) A single Duck superclass with species-specific subclasses ✓ Correct
C) A database-driven rules engine for duck behavior
D) A functional pipeline of duck actions

Rationale: The initial design used one Duck superclass from which all duck types inherited.

Q3.
What new feature did the SimUDuck executives request for the simulator?

A) Ducks that can dance
B) Ducks that can dive underwater
C) Ducks that can fly ✓ Correct
D) Ducks that can migrate seasonally

Rationale: The chapter explicitly says the executives decided the simulator needed flying ducks.

Q4.
What problem occurred when flying behavior was added to the Duck superclass?

A) Mallard ducks stopped quacking
B) Rubber ducks incorrectly gained the ability to fly ✓ Correct
C) The simulator could no longer display ducks
D) All ducks lost their swimming behavior

Rationale: Adding fly() to the superclass caused inappropriate subclasses such as rubber ducks to fly.

Q5.

The chapter describes the flying rubber duck issue as what kind of effect?

A) A compile-time optimization

B) A localized update with a non-local side effect ✓ Correct

C) A concurrency deadlock

D) A database transaction anomaly

Rationale: The text specifically calls it a localized update that caused a non-local side effect.

Q6.

Which approach did Joe first consider after inheritance proved problematic for fly and quack behaviors?

A) Using abstract factories

B) Using interfaces such as Flyable and Quackable ✓ Correct

C) Using a singleton Duck manager

D) Using a relational schema for behaviors

Rationale: Joe next considered having only some duck types implement Flyable and Quackable.

Q7.

Why did using interfaces like Flyable and Quackable fail to fully solve the SimUDuck design problem?

A) Interfaces prevented polymorphism

B) Interfaces required database persistence

C) Interfaces typically provided no implementation code, reducing reuse ✓ Correct

D) Interfaces forced all ducks to share the same constructor

Rationale: The chapter notes that Java interfaces typically have no implementation code, so behavior code reuse is lost.

Q8.

According to the chapter, what is the one constant in software development?

A) Inheritance

B) Compilation

C) Change ✓ Correct

D) Documentation

Rationale: The text emphasizes that applications must grow and change over time or they die.

Q9.

Which design principle is introduced to address changing behavior in the Duck example?

A) Minimize object creation

B) Take what varies and encapsulate it ✓ Correct

C) Always favor subclassing

D) Avoid polymorphism

Rationale: The first principle presented is to encapsulate the parts of the code that vary.

Q10.

In the Duck example, which two behaviors were identified as varying across duck types?

A) display() and swim()
B) eat() and sleep()
C) fly() and quack() ✓ Correct
D) draw() and move()

Rationale: The chapter identifies fly() and quack() as the behaviors that vary across ducks.

Q11.

What names are given to the two behavior interfaces introduced in the redesigned SimUDuck application?

A) DuckBehavior and BirdBehavior
B) MoveBehavior and SoundBehavior
C) FlyBehavior and QuackBehavior ✓ Correct
D) ActionBehavior and VoiceBehavior

Rationale: The redesign introduces FlyBehavior and QuackBehavior as the behavior supertypes.

Q12.

In the redesigned architecture, who implements the flying and quacking behavior interfaces?

A) Only the Duck superclass
B) Only utility classes
C) Behavior classes dedicated to those actions ✓ Correct
D) Every duck subclass directly

Rationale: The chapter states that separate behavior classes implement the behavior interfaces, not the duck classes themselves.

Q13.

The phrase "Program to an interface" is clarified in the chapter as meaning what?

A) Program only in Java interfaces
B) Program to a supertype ✓ Correct
C) Program to the database schema
D) Program without using classes

Rationale: The chapter explicitly reframes the phrase as "Program to a supertype."

Q14.

What is the main purpose of programming to a supertype?

A) To ensure all objects are immutable
B) To lock code to one implementation
C) To exploit polymorphism and avoid dependence on concrete types ✓ Correct
D) To eliminate constructors

Rationale: Programming to a supertype allows the actual runtime object to vary through polymorphism.

Q15.
In the Animal example, which pair of classes are presented as concrete implementations of the abstract class Animal?

A) Wolf and Fox
B) Dog and Cat ✓ Correct
C) Duck and Goose
D) Lion and Tiger

Rationale: The chapter uses Dog and Cat as concrete implementations of Animal.

Q16.
What are the names of the two instance variables added to the Duck class in the redesign?

A) flyType and quackType
B) wingMode and soundMode
C) flyBehavior and quackBehavior ✓ Correct
D) doFly and doQuack

Rationale: The Duck class gains FlyBehavior and QuackBehavior references named flyBehavior and quackBehavior.

Q17.
Which methods replace the original fly() and quack() methods in the redesigned Duck class?

A) runFly() and runQuack()
B) performFly() and performQuack() ✓ Correct
C) executeFly() and executeQuack()
D) startFly() and startQuack()

Rationale: The Duck class delegates through performFly() and performQuack().

Q18.
How does performQuack() work in the redesigned Duck class?

A) It casts the duck to MallardDuck and calls quack()
B) It directly prints a hardcoded sound
C) It asks the quackBehavior object to quack ✓ Correct
D) It invokes a static method on QuackBehavior

Rationale: The delegated method simply calls quack() on the object referenced by quackBehavior.

Q19.
Which concrete flying behavior is assigned to a MallardDuck in its constructor?

A) FlyNoWay
B) FlyRocketPowered
C) FlyWithWings ✓ Correct
D) GlideOnly

Rationale: MallardDuck is initialized with FlyWithWings as its flying behavior.

Q20.
Which concrete quacking behavior is assigned to a MallardDuck in its constructor?

A) MuteQuack
B) Squeak
C) Whistle
D) Quack ✓ Correct

Rationale: MallardDuck is initialized with Quack as its quacking behavior.

Q21.
Which class represents a duck that initially cannot fly in the runtime behavior example?

A) RubberDuck
B) ModelDuck ✓ Correct
C) RedheadDuck
D) DecoyDuck

Rationale: The chapter introduces ModelDuck for the example of changing behavior dynamically.

Q22.
Which class is created to add rocket-powered flying to the simulator?

A) TurboFly
B) RocketDuck
C) FlyRocketPowered ✓ Correct
D) FlyWithEngine

Rationale: The text specifically says to create FlyRocketPowered implementing FlyBehavior.

Q23.
How can a duck's behavior be changed at runtime in the redesigned system?

A) By recompiling the subclass
B) By editing the Duck superclass
C) By calling a setter method for the behavior ✓ Correct
D) By replacing the inheritance hierarchy

Rationale: The chapter notes that runtime changes are made by calling the duck's setter method for that behavior.

Q24.
What term does the chapter use to describe the different flying and quacking implementations as a group?

A) A family of algorithms ✓ Correct
B) A set of transactions
C) A persistence layer
D) A type-safe enumeration

Rationale: The chapter reframes the behaviors as a family of algorithms.

Q25.
Which relationship best describes how a Duck uses FlyBehavior and QuackBehavior in the redesigned model?

A) IS-A
B) HAS-A ✓ Correct
C) DEPENDS-ONLY-AT-COMPILE-TIME
D) IMPLEMENTS

Rationale: Each duck has a FlyBehavior and a QuackBehavior object.

Q26.
What OO technique is being used when a Duck contains behavior objects rather than inheriting the behavior directly?

A) Reflection
B) Composition ✓ Correct
C) Serialization
D) Overloading

Rationale: The chapter identifies this arrangement as composition.

Q27.
What is the third design principle introduced in the chapter?

A) Prefer static methods over instance methods
B) Favor composition over inheritance ✓ Correct
C) Use inheritance for all reusable behavior
D) Avoid interfaces in domain models

Rationale: The chapter explicitly states the principle as favor composition over inheritance.

Q28.
Which design pattern is formally identified as the solution used to rework SimUDuck?

A) Observer
B) Factory Method
C) Decorator
D) Strategy ✓ Correct

Rationale: The chapter states that the redesign applies the Strategy Pattern.

Q29.
Which statement best matches the chapter's formal definition of the Strategy Pattern?

A) It centralizes object creation in subclasses
B) It defines a family of algorithms, encapsulates each one, and makes them interchangeable ✓ Correct
C) It ensures one instance of a class per JVM
D) It separates construction from representation

Rationale: That sentence is the formal definition provided in the chapter.

Q30.

What does the chapter say design patterns provide beyond code?

A) A compiler plugin
B) A shared vocabulary ✓ Correct
C) A database abstraction layer
D) A fixed implementation library

Rationale: One major benefit emphasized is a shared vocabulary for communicating design ideas.

Q31.

According to the chapter, what does using a pattern name communicate to other developers?

A) Only the exact class names used
B) Only the sequence of method calls
C) A set of qualities, characteristics, and constraints ✓ Correct
D) The final compiled bytecode structure

Rationale: Pattern names communicate more than labels; they convey design qualities, characteristics, and constraints.

Q32.

Why are shared pattern vocabularies described as powerful?

A) They eliminate the need for source control
B) They allow teams to communicate design intent efficiently ✓ Correct
C) They guarantee bug-free code
D) They replace architecture documentation entirely

Rationale: The chapter says they help developers say more with less and reduce misunderstanding.

Q33.

What does the chapter say patterns do before they go into your code?

A) They go into your database
B) They go into your brain ✓ Correct
C) They go into a code generator
D) They go into UML tools

Rationale: The text explicitly says design patterns first go into your brain.

Q34.

How are design patterns distinguished from libraries and frameworks in the chapter?

A) Patterns are executable components, while libraries are not
B) Patterns are lower-level than APIs
C) Patterns are general design solutions rather than plug-in implementations ✓ Correct
D) Patterns can only be used in Java

Rationale: The chapter explains that patterns guide structure, whereas libraries and frameworks provide concrete implementations.

Q35.

Which statement about design patterns is supported by the chapter?

A) Patterns are invented from scratch for each project

B) Patterns are discovered from proven solutions ✓ Correct

C) Patterns eliminate all design tradeoffs

D) Patterns are a substitute for OO principles

Rationale: The bullet points state that patterns are discovered, not invented.

Q36.

Which quality is NOT listed among the characteristics of good OO designs in the chapter?

A) Reusable

B) Extensible

C) Maintainable

D) Hardware-dependent ✓ Correct

Rationale: The chapter identifies reusable, extensible, and maintainable as good OO design qualities.

Q37.

In the chapter's discussion, why is knowing OO basics alone insufficient?

A) Because OO basics prevent abstraction

B) Because flexible and maintainable OO design is not always obvious ✓ Correct

C) Because patterns replace encapsulation

D) Because inheritance is never useful

Rationale: The chapter explains that building flexible, reusable, maintainable systems is not automatically obvious from OO basics alone.

Q38.

According to the chapter, most patterns and principles primarily address what issue?

A) Graphical rendering

B) Network latency

C) Change in software ✓ Correct

D) Database indexing

Rationale: The bullet points state that most patterns and principles address issues of change in software.

Q39.

What does the chapter say most patterns allow?

A) Complete elimination of inheritance

B) Some part of a system to vary independently of all other parts ✓ Correct

C) Automatic generation of UML diagrams

D) Removal of all abstract classes

Rationale: The chapter emphasizes that most patterns enable one part of a system to vary independently.

Q40.
In the chapter's Q&A, why is Duck not made an interface in this example?

A) Because interfaces cannot support polymorphism
B) Because the design still benefits from shared inherited properties and methods ✓ Correct
C) Because Java forbids behavior interfaces in the same package
D) Because abstract classes cannot contain methods

Rationale: The text says Duck remains a class so specific ducks can inherit common properties and methods.

Q41.
Which of the following is given as a possible non-duck user of Quack behavior?

A) A weather sensor
B) A duck call device ✓ Correct
C) A fish tank controller
D) A game scoreboard

Rationale: The chapter suggests a duck call as an example of something that might use Quack behavior.

Q42.
In the action adventure game design puzzle, which element is the abstract class?

A) WeaponBehavior
B) SwordBehavior
C) Character ✓ Correct
D) King

Rationale: The solution states that Character is the abstract class for the game characters.

Q43.
In the action adventure game design puzzle, which element is the interface?

A) Character
B) WeaponBehavior ✓ Correct
C) Knight
D) Troll

Rationale: The solution identifies WeaponBehavior as the interface implemented by weapon classes.

Q44.
In the action adventure game design puzzle, where does setWeapon() belong?

A) In each concrete weapon class
B) In the Character superclass ✓ Correct
C) Only in the King class
D) In the WeaponBehavior interface

Rationale: The chapter's solution places setWeapon() in the Character superclass.

Q45.

Which statement best captures why composition improved SimUDuck flexibility?

A) It prevented use of interfaces

B) It allowed behavior objects to be replaced as long as they shared the correct supertype ✓ Correct

C) It forced all ducks to use the same quack implementation

D) It eliminated the need for constructors

Rationale: Composition made runtime replacement of behaviors possible through common behavior interfaces.

Q46.

Which disadvantage of using inheritance for Duck behavior is explicitly identified in the chapter?

A) It makes runtime behavior changes difficult ✓ Correct

B) It prevents ducks from swimming

C) It removes the need for testing

D) It guarantees duplicated constructors only

Rationale: The chapter lists runtime behavior changes as difficult when behavior is supplied through inheritance.

Q47.

Which disadvantage of subclass-based behavior is highlighted by the flying rubber duck example?

A) Inheritance eliminates polymorphism

B) Changes can unintentionally affect other ducks ✓ Correct

C) Interfaces cannot be added later

D) Subclasses cannot override behavior

Rationale: The example demonstrates that modifying shared inherited behavior can create unintended effects in other subclasses.

Q48.

Which concrete quacking behavior class represents silence?

A) Quack

B) Squeak

C) MuteQuack ✓ Correct

D) QuietDuck

Rationale: MuteQuack is the concrete QuackBehavior implementation for no quack sound.

Q49.

Which concrete quacking behavior class is associated with rubber ducks?

A) Quack

B) Squeak ✓ Correct

C) MuteQuack

D) EchoQuack

Rationale: The chapter identifies squeaking as a distinct quack behavior implementation used for rubber ducks.

Q50.

Which statement best summarizes the chapter's view of design patterns?

A) They are fixed code modules copied directly into applications

B) They are general, time-tested solutions adapted to specific design problems ✓ Correct

C) They are alternatives to polymorphism and encapsulation

D) They are useful only when OO basics are absent

Rationale: The chapter describes patterns as proven, general solutions that are adapted to a particular application context.

Q51.

A software team maintains a simulation in which several product types share a superclass. After adding a new method to the superclass, unrelated subclasses begin exhibiting inappropriate behavior. Which design issue does this scenario best illustrate?

A) Localized updates can create non-local side effects when variable behavior is placed in a superclass ✓ Correct

B) Abstract classes prevent subclasses from customizing inherited methods

C) Polymorphism requires every subclass to support identical runtime behavior

D) Interfaces automatically eliminate maintenance problems caused by inheritance

Rationale: Putting changing behavior in the superclass can unintentionally affect subclasses that should not share that behavior.

Q52.

A developer wants only some duck types in SimUDuck to fly, but also wants to avoid duplicating flying code across many subclasses. Which redesign most directly addresses both concerns?

A) Move fly() into every concrete Duck subclass

B) Create a FlyBehavior supertype with separate concrete flying implementations used by ducks ✓ Correct

C) Keep fly() in Duck and override it with empty methods where needed

D) Convert every Duck subclass into an interface

Rationale: Encapsulating flying in separate behavior classes allows only appropriate ducks to fly while preserving code reuse.

Q53.

A team replaces inherited fly() and quack() methods with behavior objects assigned to each duck. In daily operations, what is the primary architectural benefit of this change?

A) Each duck can inherit multiple superclasses safely

B) Behavior changes can be made independently of the Duck hierarchy ✓ Correct

C) The need for polymorphism is removed from the design

D) Concrete duck classes no longer need constructors

Rationale: Separating variable behaviors into their own classes lets those behaviors evolve without changing the Duck hierarchy.

Q54.

A project manager asks why the team should invest in design patterns instead of simply reusing old source code. Based on the chapter, what is the best response?

A) Patterns primarily provide faster compilers and stronger type checking

B) Patterns enable experience reuse by applying proven design solutions to recurring problems ✓ Correct

C) Patterns eliminate the need to understand object-oriented principles

D) Patterns are packaged code libraries that can be linked directly into applications

Rationale: Design patterns capture proven design experience that can be reused across many contexts.

Q55.

A legacy application changes frequently because users request new features every quarter. Which principle from the chapter should guide the next redesign step?

A) Keep all related behavior in one superclass for consistency

B) Encapsulate what varies so changes do not ripple through stable code ✓ Correct

C) Prefer direct instantiation of concrete classes for predictability

D) Replace all abstract types with concrete implementations

Rationale: The chapter emphasizes isolating changing aspects so the rest of the system remains unaffected.

Q56.

A developer proposes having all duck subclasses implement Flyable and Quackable directly. Why is this still operationally weak for a changing product line?

A) It prevents any duck from having both behaviors simultaneously

B) It removes inheritance but still leads to duplicated behavior implementations and maintenance overhead ✓ Correct

C) It requires every behavior to be static rather than dynamic

D) It makes interfaces impossible to test in Java

Rationale: Directly implementing behavior interfaces in each subclass avoids inappropriate behavior but sacrifices code reuse and increases maintenance.

Q57.

In a code review, a senior engineer says, 'Program to an interface, not an implementation.' In the SimUDuck redesign, what does this most practically mean?

A) Declare behavior variables using supertypes so different concrete behaviors can be substituted ✓ Correct

B) Avoid using constructors when creating any object

C) Replace all classes with Java interface declarations

D) Write only abstract methods and no concrete code

Rationale: Using supertype references such as FlyBehavior allows interchangeable concrete implementations.

Q58.

A team wants a MallardDuck to use normal flying initially but switch to rocket-powered flying during a demo. Which design choice makes this possible with minimal code impact?

A) Hardcode FlyWithWings inside performFly()

B) Store flying behavior in a FlyBehavior reference and expose a setter method ✓ Correct

C) Override fly() in every duck subclass with conditional statements

D) Use only static methods for all flying algorithms

Rationale: A behavior reference plus a setter lets the duck swap concrete flying strategies at runtime.

Q59.

A training lead asks junior developers why composition was preferred over inheritance in the duck example. Which answer is most accurate?

A) Composition guarantees fewer classes in the design

B) Composition lets ducks gain behavior by HAS-A relationships that can vary independently ✓ Correct

C) Composition removes the need for abstract classes and interfaces

D) Composition ensures all objects share identical behavior implementations

Rationale: The chapter shows that ducks have behavior objects, enabling flexibility beyond fixed inherited behavior.

Q60.

A new requirement introduces a duck call device that makes duck sounds but is not itself a duck. Which approach best follows the chapter's design guidance?

A) Subclass Duck so the device can inherit quack()

B) Implement QuackBehavior in a separate device class or compose the device with a quacking behavior ✓ Correct

C) Add duck call logic to the Duck superclass for reuse

D) Create a new Duck subtype with no display() method

Rationale: Because quacking is encapsulated as a behavior, non-duck objects can reuse it without inheriting from Duck.

Q61.

A developer says, 'Since MallardDuck uses new Quack() in its constructor, we are still tied to implementations.' Which evaluation best matches the chapter?

A) The concern is valid, but the design still gains flexibility because the stored reference is the behavior supertype ✓ Correct

B) The concern is invalid because constructors never affect flexibility

C) The concern is irrelevant because inheritance is more flexible than composition

D) The concern proves the Strategy Pattern cannot support runtime changes

Rationale: Although initialization uses concrete classes, the polymorphic behavior reference still permits later substitution.

Q62.
A hospital IT team is redesigning a rules engine where calculation methods differ by state and may change often. Which lesson from the duck example is most transferable?

A) Put every tax calculation method in a shared superclass to simplify updates
B) Encapsulate each calculation algorithm so it can vary independently from the clients using it ✓ Correct
C) Avoid interfaces because they reduce code reuse
D) Represent each state as a subclass with duplicated calculation code

Rationale: The chapter explicitly generalizes the duck behavior approach to families of algorithms such as state tax calculations.

Q63.
In the reworked SimUDuck design, what is the role of performQuack()?

A) It overrides all quack behavior in subclasses
B) It delegates quacking to the current QuackBehavior object ✓ Correct
C) It selects a quack algorithm through inheritance at compile time
D) It stores the quack sound directly in the Duck superclass

Rationale: performQuack() forwards the request to the composed behavior object.

Q64.
A team introducing design patterns into its workflow wants a concise reason they improve design discussions. Which statement best fits the chapter?

A) Patterns replace the need for architecture meetings
B) Patterns provide a shared vocabulary that communicates design qualities and constraints efficiently ✓ Correct
C) Patterns are mainly useful for naming variables consistently
D) Patterns prevent disagreements about implementation details by eliminating tradeoffs

Rationale: The chapter stresses that pattern names convey a rich set of design implications and improve communication.

Q65.
A developer is choosing between keeping fly() in Duck or extracting it. Which observation most strongly supports extraction?

A) display() methods differ among ducks
B) Flying behavior varies across duck types and is likely to change over time ✓ Correct
C) All ducks are instantiated through constructors
D) Quacking and flying must always occur together

Rationale: The principle is to separate the aspects that vary, and flying is identified as such a varying behavior.

Q66.
A software architect asks whether design patterns are the same as frameworks. Which response aligns with the chapter?

A) Yes, both are drop-in code packages for direct compilation
B) No, patterns are higher-level design solutions, while frameworks and libraries are concrete implementations ✓ Correct
C) Yes, patterns are simply smaller frameworks
D) No, frameworks never use patterns internally

Rationale: The chapter distinguishes patterns as design guidance rather than code artifacts like libraries or frameworks.

Q67.
A team wants to make future changes less risky after repeated regressions from superclass edits. Which redesign outcome is most desirable according to the chapter?

A) Increase dependence on subclass overrides
B) Reduce unintended consequences by isolating changing behavior behind behavior interfaces ✓ Correct
C) Place all methods in one abstract base class for visibility
D) Use concrete types everywhere to simplify debugging

Rationale: Encapsulating variable behavior behind interfaces limits ripple effects and supports safer change.

Q68.
A developer reviewing the Strategy Pattern definition asks what is meant by a 'family of algorithms.' In the duck example, which pair best represents such families?

A) display() and swim()
B) Duck constructors and test classes
C) Different flying behaviors and different quacking behaviors ✓ Correct
D) Superclass fields and subclass names

Rationale: The chapter frames fly and quack implementations as interchangeable algorithm families.

Q69.
A product owner expects frequent requirement changes every six months. Which design approach is most consistent with the chapter's advice for this environment?

A) Optimize for initial code reuse through inheritance alone
B) Design stable and variable parts separately so anticipated changes affect fewer classes ✓ Correct
C) Delay all refactoring until defects appear in production
D) Use interfaces only when no behavior implementation is needed

Rationale: The chapter emphasizes anticipating change and separating volatile behavior from stable structure.

Q70.
In the action adventure game puzzle, characters can switch weapons during play. Which design move mirrors the duck solution most closely?

A) Make every weapon a subclass of Character
B) Give Character a WeaponBehavior reference and a setWeapon() method ✓ Correct
C) Put all weapon logic in the abstract Character superclass
D) Require each concrete character to implement every weapon algorithm

Rationale: This mirrors Strategy by composing characters with interchangeable weapon behaviors selected at runtime.

Q71.
A team member argues that knowing encapsulation, inheritance, and polymorphism is enough to guarantee good object-oriented design. According to the chapter, what is the best rebuttal?

A) OO basics automatically lead to maintainable systems only in small applications
B) Good OO design also depends on learned, often non-obvious patterns and principles for handling change ✓ Correct
C) Design patterns are mainly syntax shortcuts for OO languages
D) Inheritance is the only advanced concept needed beyond the basics

Rationale: The chapter states that flexible, maintainable OO design is not automatic from the basics alone.

Q72.
A developer must justify adding setter methods for behaviors to the Duck class. What is the strongest rationale?

A) Setter methods reduce the number of concrete behavior classes needed
B) Setter methods allow runtime substitution of behavior implementations without changing duck subclasses ✓ Correct
C) Setter methods eliminate the need for constructors in concrete ducks
D) Setter methods ensure all ducks share identical behavior objects

Rationale: Behavior setters support dynamic behavior changes, a key benefit of the redesigned approach.

Q73.
A codebase currently declares variables as specific concrete classes, such as Dog pet = new Dog(). Which alternative best demonstrates programming to a supertype?

A) Animal pet = new Dog() ✓ Correct
B) Dog pet = new Animal()
C) Object pet = new Object()
D) Animal pet = new Animal()

Rationale: Declaring the variable as the supertype allows different concrete Animal implementations to be substituted.

Q74.
A lead developer wants to explain why patterns should be learned even though they do not provide ready-made source code. Which explanation is best?

A) Patterns are executable templates generated by the compiler
B) Patterns offer general, time-tested solutions that must be adapted to a specific application ✓ Correct
C) Patterns eliminate the need for application-specific design decisions
D) Patterns are mainly historical examples with little current value

Rationale: The chapter emphasizes that patterns are reusable design solutions, not plug-in code.

Q75.
A maintenance team notices that changing one quacking implementation currently requires edits in many duck subclasses. Which redesign would most directly reduce this operational burden?

A) Move quack code into a single QuackBehavior implementation hierarchy used by composition ✓ Correct
B) Duplicate the latest quack code across all subclasses for consistency
C) Replace quack behavior with subclass-specific constants
D) Convert Duck into a final class

Rationale: Centralizing quack algorithms in behavior classes allows one implementation change to serve many ducks.

Q76.
A developer asks why the chapter says 'program to an interface' really means 'program to a supertype.' What is the best interpretation?

A) Only Java interfaces can support polymorphism
B) The important idea is using an abstract type so clients depend on capabilities rather than concrete classes ✓ Correct
C) Supertypes should never contain any methods
D) Concrete classes are forbidden in object-oriented design

Rationale: The principle is broader than Java interfaces and focuses on polymorphic use of abstract types.

Q77.
A company is debating whether to form a design patterns study group. Which organizational benefit is most strongly supported by the chapter?

A) It guarantees all projects will use identical architectures
B) It helps build a shared community and vocabulary that speeds communication and onboarding ✓ Correct
C) It removes the need for senior developer oversight
D) It replaces code reviews with pattern reviews

Rationale: The chapter highlights how shared pattern vocabulary improves team communication and helps junior developers learn faster.

Q78.

A developer adds FlyRocketPowered as a new class implementing FlyBehavior. What does this change demonstrate about the revised architecture?

A) New behavior can be added by extending Duck without touching existing behavior classes

B) New algorithms can be introduced with minimal effect on existing ducks because behaviors are encapsulated ✓ Correct

C) Every new flying style requires a new Duck superclass

D) Runtime flexibility is reduced when concrete behavior classes increase

Rationale: Adding a new behavior implementation shows that the algorithm family can expand independently of client ducks.

Q79.

A system analyst asks what relationship exists between Duck and FlyBehavior in the revised design. Which answer is correct?

A) Duck IMPLEMENTS FlyBehavior

B) Duck EXTENDS FlyBehavior

C) Duck HAS-A FlyBehavior ✓ Correct

D) Duck and FlyBehavior are unrelated

Rationale: The design uses composition, so each duck has a FlyBehavior reference.

Q80.

During a design meeting, two developers quickly communicate that a subsystem uses Strategy. What information is most likely conveyed beyond the pattern name itself?

A) The subsystem avoids all use of inheritance and abstract classes

B) The subsystem encapsulates interchangeable algorithms behind a common interface and can vary them independently ✓ Correct

C) The subsystem uses only runtime reflection to select methods

D) The subsystem stores all logic in a single controller object

Rationale: Saying 'Strategy' conveys encapsulated interchangeable behaviors and independent variation from clients.

Q81.

A developer is comparing two designs for changing behavior: subclass overrides versus composed behavior objects. Which operational advantage of composed behavior objects is emphasized in the chapter?

A) They remove the need to instantiate objects at runtime

B) They support changing behavior dynamically after object creation ✓ Correct

C) They guarantee lower memory use than inheritance

D) They eliminate all constructor logic

Rationale: The chapter specifically shows runtime behavior changes through setter methods on composed behavior references.

Q82.
A team is tempted to refactor Duck into an interface because behaviors have been extracted. According to the chapter, why is keeping Duck as a class still useful here?

A) A class can still provide shared properties and methods common to all ducks ✓ Correct
B) Interfaces cannot participate in polymorphism
C) Classes prevent the use of composition
D) Only classes can contain references to behavior objects

Rationale: The chapter notes that Duck can remain a class to preserve common state and behavior among duck types.

Q83.
A business stakeholder asks why software design must focus so much on change. Which chapter-based answer is strongest?

A) Because most software is rewritten from scratch every release
B) Because applications must grow and adapt over time or they become obsolete ✓ Correct
C) Because change only affects user interfaces, not architecture
D) Because stable requirements are incompatible with object-oriented programming

Rationale: The chapter states that growth and change are constants in software development.

Q84.
A developer wants to know whether a behavior class can legitimately have state. Which answer best reflects the chapter?

A) No, behavior classes must contain methods only
B) Yes, behavior classes may include state such as speed or altitude parameters for flying ✓ Correct
C) No, state belongs exclusively in subclasses of Duck
D) Yes, but only if the behavior class extends Duck

Rationale: The chapter explicitly notes that behavior objects can still have instance variables representing behavior attributes.

Q85.
A design review identifies duplicated quack logic in multiple classes outside the Duck hierarchy. Which lesson from the chapter most directly applies?

A) Move all shared code into a utility class of static methods
B) Treat quacking as a reusable behavior family that can be composed into different clients ✓ Correct
C) Convert every client into a Duck subclass
D) Use inheritance to force all sound-producing objects into one tree

Rationale: Encapsulated behavior families can be reused by ducks and non-duck clients alike through composition.

Q86.
A team is selecting a pattern for a module where one object must choose among multiple interchangeable algorithms at runtime. Which pattern from the chapter is the best fit?

A) Singleton
B) Observer
C) Strategy ✓ Correct
D) Decorator
Rationale: Strategy is defined as encapsulating interchangeable algorithms that vary independently from clients.

Q87.
A programmer says, 'If patterns are not code libraries, then how do I actually use them?' Which answer best matches the chapter?

A) Import them as packages and subclass their default implementations
B) Learn the pattern concepts, then adapt their structures to your own classes and objects ✓ Correct
C) Generate them automatically from UML tools without design decisions
D) Replace frameworks with patterns to reduce dependencies
Rationale: Patterns are applied mentally as design solutions and then implemented within the application's own structure.

Q88.
A product team wants to minimize misunderstandings during architecture discussions. Which chapter-based practice would most help?

A) Avoid all pattern terminology so meetings stay implementation-focused
B) Use shared pattern vocabulary so design intent can be communicated compactly and accurately ✓ Correct
C) Require every design discussion to include code-level examples immediately
D) Standardize on one inheritance hierarchy for all projects
Rationale: The chapter emphasizes that shared pattern vocabulary improves communication and keeps discussion at the design level.

Q89.
A developer is deciding whether to instantiate a behavior implementation at compile time or assign it later. Which approach offers the greatest flexibility in the chapter's model?

A) Hardcode behavior methods directly in Duck
B) Assign a concrete behavior object to a supertype reference at runtime ✓ Correct
C) Require all ducks to share one global behavior instance permanently
D) Use only subclass overrides selected by the compiler
Rationale: Runtime assignment to a behavior supertype reference maximizes interchangeability and flexibility.

Q90.
A quality improvement team is documenting why inheritance alone failed in SimUDuck. Which item should be included as a documented disadvantage from the chapter?

A) Inheritance prevents all code reuse
B) Changing inherited behavior can unintentionally affect other subclasses ✓ Correct
C) Inheritance makes object creation impossible at runtime
D) Inheritance removes the need for interfaces

Rationale: The chapter explicitly notes that superclass changes can produce unintended effects in unrelated subclasses.

Q91.
A developer needs to explain the difference between HAS-A and IS-A using the revised duck design. Which statement is correct?

A) MallardDuck IS-A FlyBehavior because it can fly
B) Duck HAS-A QuackBehavior because quacking is delegated to a composed object ✓ Correct
C) QuackBehavior IS-A Duck because ducks use quacking
D) FlyWithWings HAS-A Duck because it is assigned to one

Rationale: The revised design uses composition, so ducks have behavior objects rather than being those behaviors.

Q92.
A project sponsor asks what practical value patterns add beyond naming concepts elegantly. Which answer is most accurate?

A) Patterns guarantee defect-free code generation
B) Patterns capture proven solutions that improve maintainability, extensibility, and reuse of design experience ✓ Correct
C) Patterns replace the need for testing in flexible systems
D) Patterns are useful only in textbook examples

Rationale: The chapter presents patterns as proven design experience that supports good OO qualities.

Q93.
A team is redesigning a simulator and identifies fly() and quack() as the only frequently changing parts. What is the most appropriate next step?

A) Refactor display() first because all methods should vary together
B) Extract fly and quack into separate behavior abstractions while leaving the stable Duck structure mostly intact ✓ Correct
C) Eliminate Duck subclasses entirely
D) Merge all changing and stable code into one behavior manager class

Rationale: The chapter recommends pulling out the varying behaviors while preserving the stable parts of Duck.

Q94.

A software engineer asks whether patterns must be applied only after code has already become difficult to maintain. Which response best reflects the chapter?

A) Yes, patterns are strictly for late-stage refactoring

B) No, patterns and principles can be applied during design or later when change points are recognized ✓ Correct

C) Yes, early use of patterns is considered overengineering in all cases

D) No, but only frameworks can be applied early

Rationale: The chapter states that developers can anticipate likely variation and build flexibility in from the start.

Q95.

A development manager wants an example of a non-obvious insight conveyed by the phrase 'favor composition over inheritance.' Which example from the chapter best supports that principle?

A) Give each duck behavior objects for flying and quacking instead of inheriting those implementations ✓ Correct

B) Place all duck logic in one superclass to reduce the class count

C) Require every subclass to implement all behavior interfaces directly

D) Represent each algorithm as a static method in Duck

Rationale: The duck redesign demonstrates composition by giving ducks behavior objects rather than fixed inherited behavior.

Q96.

A team member asks what is meant by saying patterns go into your brain before they go into code. What is the best interpretation?

A) Patterns are memorized syntax forms that compilers recognize automatically

B) Patterns are conceptual tools you learn and then apply when structuring your own designs ✓ Correct

C) Patterns should never appear in implemented software artifacts

D) Patterns are informal and therefore not useful for professional development

Rationale: The chapter explains that patterns are mental design knowledge applied to specific coding situations.

Q97.

A developer reviewing the MiniDuckSimulator notices that performFly() does not care what specific duck object is invoking it. Why is this significant?

A) It shows that duck identity is irrelevant in object-oriented systems

B) It demonstrates delegation through a common behavior interface, enabling polymorphic substitution ✓ Correct

C) It proves constructors should not initialize behavior fields

D) It means all ducks must share the same flying implementation

Rationale: performFly() relies on the behavior interface rather than the concrete duck type, which is the essence of polymorphic delegation.

Q98.

A team is deciding how to document a subsystem that computes outputs in several interchangeable ways. Which wording best reflects the formal Strategy definition from the chapter?

A) The subsystem centralizes all algorithms in one abstract base class and locks clients to it

B) The subsystem defines a family of algorithms, encapsulates each one, and makes them interchangeable ✓ Correct

C) The subsystem duplicates algorithms across subclasses to maximize specialization

D) The subsystem hides all algorithms inside constructors to simplify APIs

Rationale: This is the chapter's formal definition of the Strategy Pattern.

Q99.

A systems architect is evaluating whether to discuss a design at the object level or the pattern level during an early planning session. According to the chapter, what is an advantage of staying at the pattern level?

A) It avoids the need to identify responsibilities among classes

B) It helps the team remain focused on design intent without dropping prematurely into implementation details ✓ Correct

C) It guarantees the final code will match the initial UML exactly

D) It removes the need for later refactoring

Rationale: The chapter notes that pattern-level discussion keeps teams 'in the design' longer and reduces premature detail debates.

Q100.

A chief engineer asks for the best overall recommendation when a system must remain reusable, extensible, and maintainable under ongoing change. Which recommendation most closely matches the chapter's closing message?

A) Rely primarily on inheritance because reuse is the main goal of OO design

B) Use proven patterns and underlying OO principles to structure systems so varying parts are isolated ✓ Correct

C) Prefer libraries over design thinking because code reuse solves architecture issues

D) Avoid abstract supertypes because they hide implementation details

Rationale: The chapter concludes that good OO design comes from applying principles and patterns that isolate change and improve maintainability.