Answers are in blue.

# For Exercises 1–10, match the activity with the phase of the object-oriented methodology.



**Computer Science Illuminated, Seventh Edition**

Nell Dale, PhD; John Lewis, PhD

**CHAPTER 9**

EXERCISES AND ANSWERS

1. **Brainstorming**
2. **Filtering**
3. **Scenarios**
4. **Responsibility algorithms**
   1. Reviewing a list of possible classes, looking for duplicates or missing classes.

B

* 1. Asking “what if” questions. C
  2. Assigning responsibilities to classes. C
  3. Generating a first approximation to the list of classes in a problem.

A

* 1. Assigning collaborators to a responsibility. C
  2. Developing algorithms for the responsibilities listed on a CRC card.

D

* 1. Output from this phase is a fully developed CRC card for all classes.

C

* 1. Output from this phase is the design ready to be translated into a program.

D

* 1. During this phase, inheritance relationships are explored. B
  2. Phase in which functional programming techniques are appropriate.

D

# For Exercises 11–24, match the question with the appropriate translation or execution system.

1. **Interpreter**
2. **Assembler**
3. **Compiler**
4. **Machine code**
   1. What translates a high-level language into machine code? C
   2. What translates a Java program into Bytecode? C
   3. What executes Bytecode? A
   4. What translates an assembly-language program? B
   5. What is the output of an assembler? D
   6. What takes input in a high-level language and directs the computer to perform the actions specified in each statement? A
   7. What executes the Java Virtual Machine? D
   8. What is used to translate a program in ALGOL? C
   9. What is used to translate a program in APL? A
   10. What is used to translate a program in COBOL? C
   11. What is used to translate a program in FORTRAN? C
   12. What is used to translate a program in Lisp? A
   13. What is used to translate a program in PROLOG? A
   14. Which translator runs the most slowly? A

# For Exercises 25–46, match the language paradigm and the language or the language description.

* + 1. **Procedural**
    2. **Functional**
    3. **Logic**
    4. **Object-oriented**
    5. **Procedural language with some object-oriented features**
    6. **Object-oriented language with some procedural features**
  1. Which paradigm most accurately describes FORTRAN? A
  2. Which paradigm most accurately describes C++? E
  3. Which paradigm most accurately describes PASCAL? A
  4. Which paradigm most accurately describes Java? F
  5. Which paradigm most accurately describes Lisp? B
  6. Which paradigm most accurately describes BASIC? A
  7. Which paradigm most accurately describes PROLOG? C
  8. Which paradigm most accurately describes SIMULA? D
  9. Which paradigm most accurately describes ALGOL? A
  10. Which paradigm most accurately describes ML? B
  11. Which paradigm most accurately describes Scheme? B
  12. Which paradigm most accurately describes Python? D
  13. Which paradigm most accurately describes C? A
  14. Which paradigm most accurately describes Smalltalk? D
  15. The dominant languages used in industry throughout the history of computing software come from which paradigm? A
  16. Which paradigm did the Japanese choose for the fifth- generation computer?

C

* 1. Which paradigm allows the programmer to express algo- rithms as a hierarchy of objects?

D

* 1. Which paradigm allows the programmer to express algo- rithms as a hierarchy of tasks?

A

* 1. Which paradigm allows the programmer to express algo- rithms as mathematical functions?

B

* 1. Which paradigm has no assignment statement? B
  2. Which paradigm uses recursion exclusively to express repetition?

B

* 1. Which paradigm has no variables? B

# Exercises 47–84 are short answer.

* 1. What is the hallmark of an assembly language?

The hallmark of an assembly language is that each assembly-language instruction is translated into one machine-language instruction.

* 1. Distinguish between an assembler and a compiler.

An assembler translates assembly-language instructions into machine code. A compiler translates high-level language instructions into machine code (or assembly code). The translation of an assembler is one to one: One statement in assembly language is translated into one statement in machine code. The translation of a compiler is one to many: One high-level language instruction is translated into many machine-language instructions.

* 1. Distinguish between a compiler and an interpreter.

The output from a compiler is a machine-language program. That program may be stored for later use or immediately exe- cuted, but the execution is a distinct process from the trans- lation. An interpreter translates and executes together. The output from an interpreter is a solution to the original problem, not a program that, when executed, gives you the solution.

* 1. Compare and contrast an assembler, a compiler, and an interpreter.

All three are translators. They differ in the complexity of the languages they translate and in the output from the trans- lator. Assemblers and compilers produce machine-language programs, which, when run, solve the original problem; interpreters produce solutions to the original problem. Assemblers translate very simple languages; compilers and interpreters can translate very complex languages.

* 1. Describe the portability provided by a compiler.

A program written in a high-level language that is compiled can be translated and run on any machine that has a com- piler for the language.

* 1. Describe the portability provided by the use of Bytecode. Bytecode is the output from (usually) a Java compiler. There is a virtual machine for which Bytecode is the machine lan- guage. A program compiled into Bytecode can be executed on any system that has a simulator for the virtual machine. The Java Virtual Machine (JVM) executed Bytecode.
  2. Describe the process of compiling and running a Java program.

A Java program is compiled into Bytecode, which can be executed on any system with a JVM.

* 1. Discuss the word *paradigm* as it relates to computing. Programming languages reflect differing views of reality, which we call paradigms. We use these views (paradigms) to classify the languages.
  2. Distinguish between imperative and declarative paradigms. An imperative language describes the actions to be taken to solve the problem. A declarative language describes the solution expected, but not the steps to be taken.
  3. What are the characteristics of the imperative paradigm? Programs describe the processes necessary to solve the problem.
  4. What are the characteristics of the functional paradigm? Programs are expressed as the evaluation of functions.
  5. What are the characteristics of the logic paradigm?

Rules of logic are used to deduce answers from facts and rules.

* 1. What are the characteristics of a declarative paradigm?

A declarative language describes the solution, not the steps to get there.

* 1. How do you ask questions in a programming language?

To ask a question in a programming language, you make an assertion. If the assertion is true, the answer is true. If the assertion is false, the answer is false.

* 1. What is a Boolean variable?

A Boolean variable is a place in memory, referenced by an identifier, that can contain true or false.

* 1. What is a Boolean expression?

A Boolean expression is a sequence of identifiers, separated by compatible operators, that evaluates to true or false.

* 1. Given Boolean variables one, two, and three*,* write an asser- tion for each of the following questions:
     1. Is one greater than both two and three? (one > two) AND (one > three)
     2. Is one greater than two, but less than three? (one > two) AND (one < three)
     3. Are all three variables greater than zero? (one > 0) AND (two > 0) AND (three > 0)
     4. Is one less than two or one less than three? (one < two) OR (one < three)
     5. Is two greater than one and three less than two? (two > one) AND (three < two)
  2. Write the operation table for Boolean operation AND.

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A AND B** |
| false | false | false |
| false | true | false |
| true | false | false |
| true | true | true |

* 1. Write the operation table for Boolean operation OR.

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A OR B** |
| false | false | false |
| false | true | true |
| true | false | true |
| true | true | true |

* 1. Write the operation table for Boolean operation NOT.

|  |  |
| --- | --- |
| **A** | **NOT A** |
| false | true |
| true | false |

* 1. What is a data type?

A data type is the description of a set of values and the basic set of operations that can be applied to values of the type.

* 1. What is strong typing?

Strong typing means that each variable is assigned a data type and only values of that type can be stored in the variable.

* 1. Define the following data types.
     1. Integer

The range of integer values that a machine can represent.

* + 1. Real

The range of rational numbers that a machine can represent.

* + 1. Character

The characters in the character set that the machine supports.

* + 1. Boolean

The values true and false.

* 1. Is the string data type an atomic data type? Justify your answer.

A string can be output like an atomic data type, but it is made up of characters, each of which can be accessed separately. Thus, you can argue either side.

* 1. If the same symbol is used for both single characters and strings, how can you distinguish between a single character and a one-character string?

If the same symbol is used, a character cannot be distin- guished from a one-character string.

* 1. What is a declaration?

A declaration is an instruction to the compiler that associates an identifier with a variable, an action, or some other entity within the language that can be given a name. The program- mer can then refer to that entity by name.

* 1. Fill in the following table showing the appropriate syntactic marker or reserved word for the language shown based on your observation of the tables in this chapter.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Language** | **Python** | **VB.NET** | **C++** | **Java** |
| Comments | # | ‘ | // | // |
| End of statement | End of line or indention | End of line or comment | ; | ; |
| Assignment statement | = | = | = | = |
| Real data type | Float | Single | float | float |
| Integer data type | Integer | Integer | int | int |
| Beginning of declaration(s) | No declarations | Dim | none | none |

* 1. How do the .WORD and .BLOCK assembler directives in the Pep/8 assembly language differ from the declarations in high-level languages?

.WORD allows us to associate an identifier with a word in storage and specify what should go into that word. .BLOCK allows us to associate an identifier with a specified number of bytes. In each case, the programmer knows the actual address associated with the identifier. Declarations in high- level languages give you the same functionality, but the pro- grammer does not need to worry about the actual address.

* 1. Distinguish between instructions to be translated and instruc- tions to the translating program.

Instructions to the translating program tell the program to associate identifiers with objects, and if the objects are data, tell the program the data type of what can be stored in that place.

* 1. Consider the following identifiers: Address, ADDRESS, AddRess, Name, NAME, NamE
     1. How many different identifiers are represented if the lan-

guage is Python? 6

* + 1. How many different identifiers are represented if the lan- guage is VB.NET?

6

* + 1. How many different identifiers are represented if the lan- guage is C++ or Java?

6

* 1. Distinguish between the definition of an object in the design phase and in the implementation phase.

An object in the design phase is an entity that has meaning within the context of the problem. An object in the imple- mentation phase is an instance of a class.

* 1. Distinguish between the definition of a class in the design phase and in the implementation phase.

A class in the design phase is a description of a group of objects with similar properties and behaviors. A class in the implementation phase is a pattern for an object.

* 1. Distinguish between a field and a method.

A field represents a property of a class. A method is a named algorithm that manipulates the data values in an object. Thus, a method represents behavior.

* 1. How can objects relate to one another?

Objects can be related by containment, inheritance, or col- laboration. An object can contain another object as a field. An object can inherit the data and behavior of another object class. An object can collaborate with an object of its own class or another class.

* 1. Discuss the differences between a top-down design and an object-oriented design.

Top-down design breaks the problem into successive levels of tasks; object-oriented design breaks the problem into suc- cessive levels of data objects.

* 1. In this chapter, we outlined a strategy for developing an object-oriented decomposition.
     1. List the four stages.

Brainstorming, filtering, scenarios, and responsibility algorithms.

* + 1. Outline the characteristics of each stage.

Brainstorming is a group problem-solving activity that involves the spontaneous contribution of ideas from all members of the group. The output from this activity is a list of possible classes.

Filtering is a group activity in which the tentative list of classes is analyzed to determine if there are duplicates, if some classes share common attributes and behaviors, and if there are classes that really do not belong in the solution.

Scenarios is a group activity that determines the responsibil- ities of the classes. They ask “what if” questions and deter- mine if all possible situations have been considered.

Responsibility algorithms are the algorithms that implement the responsibilities. This phase is where the algorithms to carry out the solution get written.

* + 1. What is the output from each of the four stages? Brainstorming: A list of possible classes.

Filtering: CRC cards for the classes that survived this stage.

Scenarios: CRC cards with responsibilities outlined and col- laborators indicated.

Responsibility algorithms: Algorithms for each of the responsibilities.

* + 1. Are the stages independent? Explain.

No, each state is not independent. The first stage produces a tentative list that is used as input to the second stage. The second stage produces a list of classes that have survived the filtering stages as input to the third stage. The third stage produces completed CRC cards that are input to the fourth stage.

* 1. Design the CRC cards for an inventory system for a car deal- ership, using brainstorming, filtering, and scenarios. Brainstorming: car, idnumber, color, date arrived at dealer- ship, date sold, name of buyer, cost

Filtering: car, date, person, list of cars, idnumber, color, cost Scenarios: CRC cards produced.

|  |  |  |  |
| --- | --- | --- | --- |
| **Class Name: Car** | **Superclass:** | | **Subclasses:** |
| **Responsibilities** | | **Collaborations** | |
| Initialize itself (name, date, color, idnumber) | | String, Date, String, String | |
| Get color | | String | |
| Get dateBought | | Date | |
| Get name | | String | |
| Get idnumber | | String | |
| Set Repair (date) | | Date | |
| GetRepairDate | | Date | |
| Sold (name, date) | | Name, Date | |
| Get dateSold | | Date | |
| Get buyer | | Name | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Class Name: Date** | **Superclass:** | | **Subclasses:** |
| **Responsibilities** | | **Collaborations** | |
| Initialize itself (month, day, year) | | int, int, int | |
| Get month | | int | |
| Get day | | int | |
| Get year | | int | |
|  | | | |
| **Class Name: Name** | **Superclass:** | | **Subclasses:** |
| **Responsibilities** | | **Collaborations** | |
| Initialize itself (first, last) | | String, String | |
| Get first | | String | |
| Get last | | String | |
|  | | | |
| **Class Name:**  **List (from library)** | **Superclass:** | | **Subclasses:** |
| **Responsibilities** | | **Collaborations** | |
| Insert (car) | | Car | |
| Delete (car) | | Car | |

* 1. Design the CRC cards for a data base for a zoo, using brain- storming, filtering, and scenarios.

Brainstorming: family name, name, date of birth, date bought, food, cageNumber, sex, date of last shots

Filtering: animal, name, date, food, cageNumber, sex Scenarios:

|  |  |  |  |
| --- | --- | --- | --- |
| **Class Name: Animal** | **Superclass:** | | **Subclasses:** |
| **Responsibilities** | | **Collaborations** | |
| Initialize itself (familyName, name, dateOfBirth, dateBought, eatsWhat, cageNumber, sex) | | String, String, Date, Date, String, String, Char | |
| Get familyName | | String | |
| Get name | | String | |
| Get food | | String | |
| Get dateOfBirth | | Date | |
| Get dateBought | | Date | |
| Put shots(Date) | | Date | |
| Get DateOfShots | | Date | |
| Get cageNumber | | String | |
| Get sex | | Char | |
|  | | | |
| **Class Name: Date** | **Superclass:** | | **Subclasses:** |
| **Responsibilities** | | **Collaborations** | |
| Initialize itself (month, day, year) | | int, int, int | |
| Get month | | int | |
| Get day | | int | |
| Get year | | int | |
|  | | | |
| **Class Name:**  **List (from library)** | **Superclass:** | | **Subclasses:** |
| **Responsibilities** | | **Collaborations** | |
| Insert (animal) | | Animal | |
| Delete (animal) | | Animal | |
| .  .  . | |  | |