Object-Oriented Design and High-Level Languages

Object-Oriented Design (Page 1)
- Object-oriented design—a problem-solving methodology that produces a solution to a problem in terms of self-contained entities called objects
- Object—a thing or entity that makes sense within the context of the problem
  - For example, a student, a car, a time, a date, etc.
  - Creating an object from a class is called instantiating (creating an “instance” of the class)

Object-Oriented Design (Page 2)
- World View of OOD—problems are solved by:
  - Isolating the objects in a problem,
  - Determining the properties and actions (responsibilities) of each of the objects, and
  - Letting the objects collaborate to solve a problem

Object-Oriented Design (Page 3)
- An analogy: You and your friend fix dinner
  - Objects: you, friend, dinner
  - Class: you and friend are people
    - People have name, eye color, ...
    - People can shop, cook, ...
  - Instance of a class: you and friend are instances of class People, you each have your own name and eye color, you each can shop and cook
  - You collaborate to fix dinner

Object-Oriented Design (Page 4)
- Class (or object class)—a description of a group of similar objects
  - Represents all objects of a kind
  - The class is a “blueprint” for the objects
- Object (instance of a class)—a concrete example of the class
  - Objects are the actual entities created from the “blueprint”
  - Represent “things” from the real world

Object-Oriented Design (Page 5)
- Classes contain fields that represent:
  - Properties (the data/attributes) (name, eye color); and ...
  - Behaviors (actions/responsibilities) (shop, cook)
- Method—a named algorithm (block of code) that defines the one of the behaviors of the object (the actions) (shop, cook), i.e. a “subprogram”

Object-Oriented Design (Page 6)
- Top-Down Design—decomposes problems into tasks
- Object-Oriented Design—decomposes problems into collaborating objects

Object-Oriented Design (Page 7)
- Steps:
  - Isolate the real-world objects in the problem
  - Abstract the objects with like properties into groups (classes)
  - Determine the responsibilities of the group in interacting with other groups

Object-Oriented Design (Page 8)
- Think of design as a mapping from real world objects to classes of objects

Object-Oriented Design (Page 9)
- Program World simulates these groups

Object-Oriented Design (Page 10)
- Responsibilities become methods in the Program World

13 **Object-Oriented Design Methodology**
- Four stages to the decomposition process
  - Brainstorming to locate possible classes
  - Filtering the classes to find duplicates or remove unnecessary ones
  - Scenarios are tried to be sure we understand collaborations
  - Responsibility algorithms are designed for all actions that classes must exhibit

14 **Brainstorming**
- A group problem-solving technique that involves the spontaneous contribution of ideas from all members of the group
  - All ideas are potential good ideas
  - Think fast and furiously first, and ponder later
  - A little humor can be a powerful force
- Brainstorming is designed to produce a list of candidate classes

15 **Filtering**
- Determine which are the core classes in the problem solution
- There may be two classes in the list that have many common attributes and behaviors
- There may be classes that really don’t belong in the problem solution

16 **Scenarios** *(Page 1)*
- Assign responsibilities to each class
- There are two types of responsibilities
  - What a class must know about itself (knowledge responsibilities)
  - What a class must be able to do (behavior responsibilities)

17 **Scenarios** *(Page 2)*
- Encapsulation—bundling of data and actions in such a way that the logical properties of data and actions are separated from implementation details
- Each class encapsulates its data but shares their values through knowledge responsibilities

18 **Responsibility Algorithms**
- Algorithms must be written for the responsibilities
  - Knowledge responsibilities usually just return the contents of one of an object’s variables
  - Action responsibilities are a little more complicated, often involving calculations

28 **Translation Process**
- A program written in a high-level language must be translated into machine code
- The machine code is then executed
- Compilers and Interpreters are software tools employed to help with the translation process

29 **Compilers**
- High-level language—a language that provides a richer (more English like) set of instructions
- Compiler—a program that translates a high-level language program into machine code

31 **Interpreters**
- Interpreter—a translating program that translates and executes the statements in sequence
  - Assembler or compiler produce machine code as output, which is then executed in a separate step
  - An interpreter translates a statement and then immediately executes the statement
  - Interpreters can be viewed as simulators

32 **Java**
- Introduced in 1996 and became instantly popular
- An object-oriented language (everything is a class)
- Portability was of primary importance
- Java is compiled into a standard machine language called Bytecode
- A software interpreter called the JVM (Java Virtual Machine) takes the Bytecode program and executes it

**Portability**
- Portability—the ability of a program to be run on different machines
- Compiler portability—a program in a standardized language can be compiled and run on any machine that has the appropriate compiler
- Bytecode portability—a program translated into Bytecode can be run on any machine that has a JVM

**Programming Language Paradigms (Page 1)**
- Imperative Paradigm—program describes the processing
- Declarative Paradigm—program describes the results
- Each of these major paradigms have distinct subparadigms

**Programming Language Paradigms (Page 2)**
- Imperative
  - Procedural
    - Characterized by sequential instructions
    - A program in which statements are grouped into a hierarchy of subprograms
    - Fortran, C, C++
  - Object-oriented model
    - Program consists of a set of objects and the interactions among the objects
    - Python, Java, Smalltalk, Simula

**Programming Language Paradigms (Page 3)**
- C++ is as a *procedural* language with some *object-oriented* features
- Java is an *object-oriented* language with some *procedural* features

**Programming Language Paradigms (Page 4)**
- Declarative
  - Functional
    - Based on the mathematical concept of a function
    - Lisp, Scheme, and ML
  - Logic
    - Based on principles of symbolic logic
    - Types of statements
      - Declares facts about objects and relationships
      - Defines rules about objects
      - Asks questions about objects
    - PROLOG

**Functionality of High-Level Languages**
- We examine procedural and object-oriented languages in the rest of this chapter by looking at the functionality provided in these languages
- We give examples in different languages to show how syntax used to provide the functionality

**Functionality of Imperative Languages**
- Sequence—executing statements in sequence until an instruction is encountered that changes this sequencing
- Selection—deciding which action to take
- Iteration (looping)—repeating an action
46 **Boolean Expressions**
- Boolean expression—a sequence of identifiers, separated by compatible operators, that evaluates to *true* or *false*
- A Boolean expression can be:
  - A Boolean variable
  - An arithmetic expression followed by a relational operator followed by an arithmetic expression
  - A Boolean expression followed by a Boolean operator followed by a Boolean expression

47 **Strong Typing**
- Data type—a description of the set of values and the basic set of operations that can be applied to values of the type
- Strong typing—the requirement that only a value of the proper type can be stored into a variable

48 **Data Types** *(Page 1)*
- Integer numbers—used to refer to a data type which represents some finite subset of the mathematical integers (numbers with no decimals)
- Real numbers—representing numbers that would be too large or too small to be represented as integers (allows for decimals)
- Characters—a data type for storing a single non-numeric data values, i.e. 'd'

49 **Data Types** *(Page 2)*
- Boolean values—a logical data type for storing data that has one of two values (true or false)
- Strings—a data type for storing a sequence of non-numeric data values, i.e. "date"

50 **Declarations**
- Declaration—a statement that associates an identifier with a variable, action, or some other entity within the language that can be given a name
  - The programmer can refer to that item by name
- Reserved word—a word in a language that has special meaning
- Case-sensitive—uppercase and lowercase letters are considered the same

52 **Assignment Statement**
- Assignment statement—an action statement (not a declaration) that says to evaluate the expression on the right-hand side of the symbol and store that value into the place named on the left-hand side
- Named constant—a location in memory, referenced by an identifier, that contains a data value that cannot be changed

53 **Input/Output Structures** *(Page 1)*
- Pseudocode algorithms used the expressions *Read or Get* and *Write or Print*
- High-level languages view input data as a stream of characters divided into lines
- Key to the processing
- The data type determines how characters are to be converted to a bit pattern (input) and how a bit pattern is to be converted to characters (output)

54 **Input/Output Structures** *(Page 2)*
*Read name, age, hourlyWage*
- *name* is a string;
- *age* is an integer;
- *hourlyWage* is a real
- The data must be a string, an integer, and a real in that order.

56 **Control Structures**
- Control structures—an instruction that determines the order in which other instructions
in a program are executed
● Sequence
● Selection
● Iteration

57 Selection Statements
- The if statement allows the program to test the state of the program variables using a Boolean expression

58 Looping Statements
- The while statement iterates through (repeats) a block of statements

59 Subprogram Statements
- We can give a section of code a name and use that name as a statement in another part of the program
- When the name is encountered, the processing in the other part of the program halts while the named code is executed

60 Nested Logic (If within a While)
Set sum to 0
Set posCount to 0
WHILE (posCount <= 10) // Test event
Read a value
IF (value > 0) // Update event?
Set posCount to posCount + 1
// Update event
Set sum to sum + value
// Statement(s) following loop

61 Nested Logic (While within a While)
Set weekCount to 1
WHILE (weekCount <= 52)
Set weekSum to 0
Set dayCount to 1
WHILE (dayCount <= 7)
Read rainfall
Set weekSum to weekSum + rainfall
Set dayCount to dayCount + 1
Write "Week " + weekCount + " total: " + weekSum
Set weekCount to weekCount +

62 Asynchronous Processing
- Asynchronous processing—not synchronized with the program's action
  ● Clicking has become a major form of computer input
  ● Mouse clicking is not within the sequence of the program
  ● A user can click a mouse at any time during the execution of a program

63 Functionality of OOPs (Page 1)
- Encapsulation—a language feature that enforces information hiding (data and actions)
  ● Classes—different meanings in different places (see earlier slide)
  ● Inheritance—a property that allows a class to inherit the data and actions of another class
- Polymorphism—the ability to handle the ambiguity of duplicate names

64 Functionality of OOPs (Page 2)
- Object class (problem-solving phase)—an entity or thing that is relevant in the context of a problem
  ● Object class (class) (problem-solving phase)—a description of a group of objects with
similar properties and behaviors
- Class (implementation phase)—a pattern for an object
- Object (implementation phase)—an instance of a class

### Class Definition (Page 1)
- A class encapsulates both data and actions
  ```java
  public class Person   // Name the class
  // Declare Class variables
  String name
  String address
  String telephone
  String email
  ```

### Class Definition (Page 2)
- // Declare Class Methods
  ```java
  public void Initialize()        // Code for Initialize
  public Person Print()    // Code for Print
  public String GetName()      // Code for Print
  RETURN Name
  public String GetAddress()    // Code for Print
  RETURN address
  public String GetEmail()      // Code for Print
  RETURN email
  public String GetTelephone()  // Code for Print
  RETURN telephone
  ```

### Class Definition (Page 3)
- Name aName = new Name()
- aName.Initialize("Frank", "Jones")
- Person aPerson = new Person()
- aPerson.Initialize(aName, address, telephone, email)
- aPerson.Print()
- Write "Name: ", aPerson.GetName().Print()
- Write "Address: ", aPerson.GetAddress()
- Write "Telephone: ", aPerson.GetTelephone()
- Write " Email: ", aPerson.GetEmail()

### Class Definition (Page 4)
- To get an object of a class, we must ask that one be created (instantiated)
- The new operator does this for us
  ```java
  Name aName = new Name()
  aName.Initialize("Frank", "Jones")
  Person aPerson = new Person()
  aPerson.Initialize(aName, address, telephone, email)
  aPerson.Print()
  Write "Name: ", aPerson.GetName().Print()
  Write "Address: ", aPerson.GetAddress()
  Write "Telephone: ", aPerson.GetTelephone()
  Write " Email: ", aPerson.GetEmail()
  ```

### Inheritance and Polymorphism (Page 1)
- Inheritance—a construct that fosters reuse by allowing an application to take an already-tested class and derive a class from it that inherits the properties the application needs
- Polymorphism—the ability of a language to have duplicate method names in an inheritance hierarchy and to apply the method that is appropriate for the object to which the method is applied

### Inheritance and Polymorphism (Page 2)
- Inheritance and polymorphism work together
- How?—they combine to allow the programmer to build useful hierarchies of classes that
can be put into a library to be reused in different applications

72 **Top-Down vs. OO Designs (Page 1)**
- Top-down Solution
  - Data structures needed in solution are determined
  - Subprograms are written to manipulate the data structures
  - Main program declares data structure
  - Main program is calls to the subprograms, passing data structures as parameters

73 **Top-Down vs. OO Designs (Page 2)**
- Object-oriented Solution
  - ADTs needed in solution are determined
  - ADTs are written only if not in library
  - Data structure is encapsulated within the class that implements the ADT
  - Main program is instructions to ADTs to perform the necessary tasks