1.3 PROGRAMIMG BASICS

- Programming : creative process
- Techniques
  - helpful when designing programming
  - Applicable any PL, not particular to Java
- Object ??

  **ob·ject [abd?ikt ｜ ?b-] n.
   (오감(五感)으로 포착할 수 있는) 물건, 물체.
  ⊙ a tiny ~ 조그마한 물건.
  a material thing that can be seen or touched.

Object-Oriented Programming: OOP

- Real world : object
  - » Ex) people, automobile, buildings, trees, shoes, ships, sealing wax, cabbage, kings…
  - » 1) Perform [redacted] ✓
    - – Each of actions has some effect on some of other objects.
  - » 2) Have [redacted](colors,size, values..)
- OOP : views a program as similarly consisting of objects that interact with one another by means of actions

Object-Oriented Programming: OOP

- A design and programming technique
- Some terminology:
  - » [redacted] usually a (particular) person, place or thing (a noun)
  - » [redacted] - an action performed by an object (a verb) ✓
  - » [redacted] - a [redacted] of similar objects (such as *automobiles*)
- Objects have both [redacted] and [redacted]
- Objects of the same class have the same data elements and methods
- Objects send and receive *messages* to invoke actions

Example of an (Object) Class

Class: Automobile

Data Items:

- manufacturer’s name
- model name
- year made
- color
- number of doors
- size of engine
- etc.

Methods:

- Define data items (specify manufacturer’s name, model, year, etc.)
- Change a data item (color, engine, etc.)
- Display data items
- Calculate cost
- etc.

Why OOP?

- Save development time (and cost) by [redacted] code
  - » once an object class is created it can be used in other applications
- [redacted] debugging
  - » classes can be tested independently
  - » reused objects have already been tested

Design Principles of OOP

Three main design principles of Object-Oriented Programming(OOP):

- 1) Encapsulation (??)
- 2) Polymorphism (??)
- 3) Inheritance (??)

Encapsulation

- *Encapsulation* means to design, produce, and describe software so that it can be easily used without knowing [redacted]
- Also known as [redacted]

An analogy:

- When you drive a car, you don't have know the details of how many cylinders the engine has or how the gasoline and air are mixed and ignited.
- Instead you only have to know [redacted] the controls.(accelerator pedal, steering wheel…)

Polymorphism

- *Polymorphism*—the same word or phrase can be mean different things in different [redacted]
  - Polymorphism : [redacted]
- Ex) Analogy: in English, **bank** can mean side of a river or a place to put money
- Ex) analog/digital dual-mode cellular phone
- In programming (??)

Polymorphism

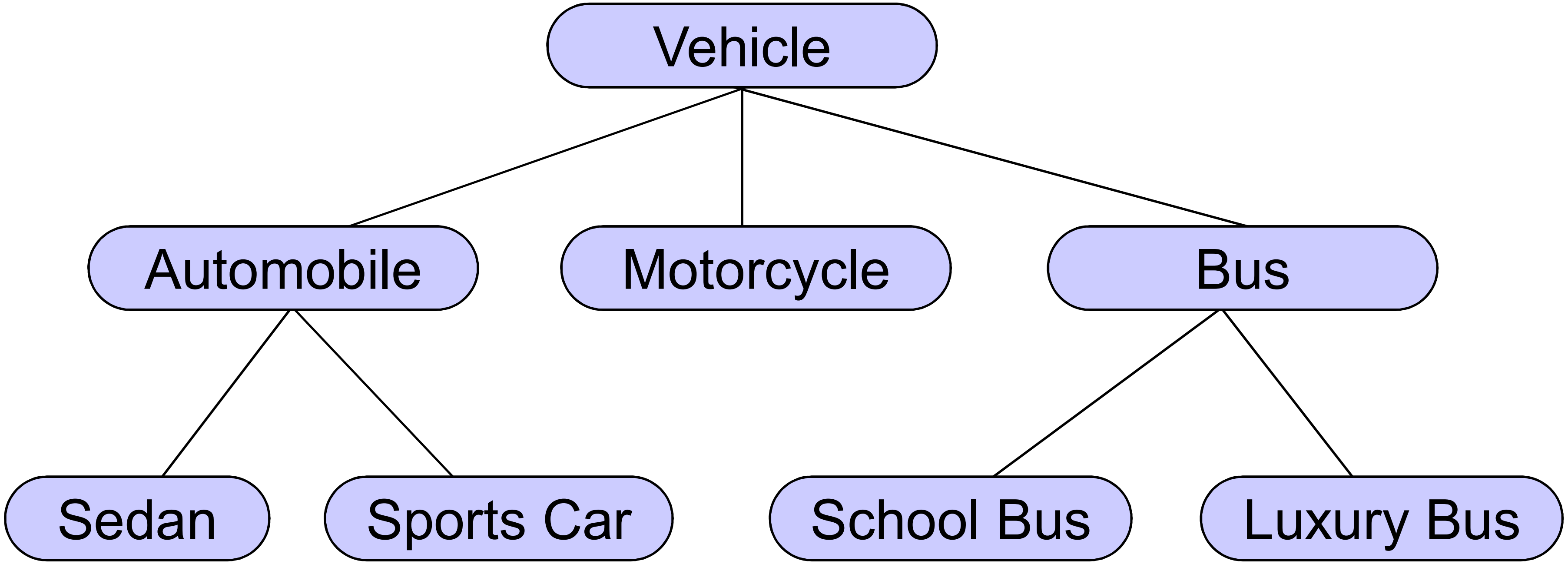
- In programming, polymorphism means that one method name, used as an instruction, can cause [redacted] actions, depending on the kind of objects that perform the actions.

- In Java, two or more classes could each have a method called **`output`**
  - » Each **`output`** method would do the right thing for the class that it was in.
  - » One **`output`** might display a number whereas a different one might display a name.

Inheritance

- *Inheritance*—a way of [redacted] classes
- Term comes from inheritance of traits like eye color, hair color, and so on.
- Classes with properties in common can be [redacted] so that their common properties are only defined [redacted]
  - » avoiding repeating the same set of programming instructions for each class.
- Display 1.4
  - » at each level, the classifications become more specialized.

Display 1.4 An Inheritance Hierarchy



What properties does each vehicle inherit from the types of vehicles above it in the diagram?

Algorithms

- *Algorithm* - a set of [redacted] (steps) for solving a problem.
  - » must be precise
  - » must be complete
- May be in a number of different formats
  - » natural language (such as English)
  - » a specific programming language
  - » a diagram, such as a flow chart
  - » [redacted] - a mix of natural and programming language

Example of an Algorithm

Algorithm that determines the total cost of a list of items:

1. Write the number 0 on the blackboard.

2. Do the following for each item on the list:

--Add the cost of the item to the number on the blackboard.

--Replace the old number on the board by this sum.

3. Announce that the answer is the number written on the board

Reusable Components

Advantages of using reusable components:

- saves time and money
- components that have been used before are often better tested and more reliable than new software

Make your classes reusable:

- [redacted]
- [redacted] classes have a better chance of being reused than [redacted] classes

Program Design Process

- Design, then code

- Design process
  - define the problem clearly
  - design objects your program needs
  - develop algorithms for the methods of objects
  - describe the algorithms, usually in pseudo_code
  - write the code
  - test the code
  - fix any errors and retest

Testing and Debugging

- Even with careful programming, your code could still contain errors and must be thoroughly tested.
- Bug—a mistake in a program
- [redacted]—fixing mistakes in a program

Types of Errors

- *Syntax*
- *Run-Time*
- *Logic*

[redacted]

- The set of [redacted] rules for a programming language is called the [redacted].
- The compiler checks your program to make sure it is a valid Java program.
- If your program is not a valid Java program, then the compiler outputs a message indicating a **<u>syntax error</u>**.

Syntax Errors

- caught by compiler ("[redacted]-time error")
- automatically found, usually the easiest to fix
- cannot run code until all syntax errors are fixed
- error message may be misleading

Example:

Misspelling a command, for example "rturn" instead of "return"

Run-Time Errors

- An [redacted] error (during run-time)
- Not always so easy to fix
- Error message may or may not be helpful
- Not detected by the [redacted].

Example:

Division by zero - if your program attempts to divide by zero it automatically terminates and prints an error message.

[redacted] Errors

Just because it compiles and runs without getting an error message does <u>not</u> mean the code is correct!

- An error in the design (the algorithm) or its implementation
  - » code compiles without errors
  - » no run-time error messages
  - » but incorrect action or data occurs during execution
- Generally the most difficult to find and fix
- Need to be alert and test thoroughly
  - » think <u>about test cases</u> and predict results ***before*** executing the code

Logic Error Examples

- Algorithm Error:
  - » **`averageOfFiveScores = SumOfScores/2`**
    (should divide by 5)

- Implementation Error:
  - » typed in wrong symbol in source code -
    `sum = a - b;`
    (should be **`sum = a + b;`**)

끝