An Overview of Data Structures with Java ( for software development )

**Software Engineering Principles and Classes**

• Designing a class consists of identifying its:

behavior: the operations that can be applied to a class object

attributes: the data that must be stored to characterize a class object

• Design a class’s behavior ( operations ) independently of implementation details of its attributes.

• Objects should carry within themselves the ability to perform their operations.

• Use an external perspective when you are *using* a class; use an internal perspective when you are *building* a class.

• A class’s behaviors should be identified before its attributes because:

- it is often not obvious what the attributes should be.

- knowing the class behavior can help with identifying the attributes.

- behaviors should be independent of any particular details of how the attributes are implemented.

• To identify a class’s attributes, go through the list of operations and identify what information reach requires. Information that is needed by several different operations is probably an attribute.

• For attributes that change over time or that differ from one object to another, define variables to store their values. Attributes that are the same for all objects and do not change can be represented as constants. Attributes determine the state of an object at a particular time.

• It is good programming practice to hide all attribute variables of a class by making them private.

• Objects share static ( or class ) constants and variables. Objects have their own copes of non - static constants and variables.

• Class operations are usually implemented using instance methods, of which five common categories are: constructors, accessors, mutators, convertors, and utilities.

• When implementing class operations with methods, it is a good practice to define methods that support output early.

• If a class has a toString() method, the existing print()and println() methods can then be used to display objects of that class, because they send the toString() message to any object they are asked to display.

• The compiler calls a constructor whenever a class object is defined using the new operator. For this reason, when building a class, always provide one or more constructors to initialize the attribute variables of the class. Otherwise, the compiler

will generate a constructor to initialize them with default values.

• A default constructor is invoked when a declaration of an object provides no initial values for the attribute variables.

• An explicit - value constructor has parameters that are used to initialize some or all of an object’s attribute variables.

• static (or class) methods and instance ( or object ) methods have the following differences:

Static methods are declared using the keyword static; instance methods are not.

Instance methods may access both static and non - static variables, constants, and methods.

Static methods may access only static variables, static constants, and static methods.

Objects share the same copy of static variables, constants, and methods, but they have their own distinct copies of non - static variables, constants and methods.

• A method that must access attribute variables must be defined as an instance method. Otherwise, it should be defined as a static method, and the information it needs should be passed to it via parameters or declared using static variables and constants.

• Since constants represent values that do not change, they should be declared as static constants.

• Explicit - value constructors are useful to construct objects that must be returned by a method.

• Accessor methods can be used to retrieve, but not modify, the value of an object’s attribute variables.

• Mutatormethods can change the values of an object’s attribute variables. They should ensure that the modified object still satisfies the class invariant.

• The keyword null should be used to initialize reference - type variables that will immediately be assigned new values.

• Factoring out code that is common to several operations and encapsulating it in a separate method can result in methods that are simpler and, therefore, easier to write and understand.

• The alias problem arises when a copy operation does not produce a completely distinct copy because the address in a reference-type variable is copied rather than the object it refers to.

• A common arrangement of items in a class is as follows:

- class constants first, so they can be quickly found

- the methods next, so it is easy to find the class’ operations; they make

up the class’ interface

- attribute variables last, so the user need not see them.

• Painting a Swing component must always be done with a method named paintComponent().

**Key Terms**

• Abstract data type ( ADT ) : a data type that specifies the logical properties without the implementation details

• Asymptotic: the study of the function f as n becomes larger and larger without bound

• Class: data and operations on that data in a single unit

• Constructor: a function within a class that guarantees that data members are initialized

• Information hiding: hides the details of the operations on the data

• Software Life Cycle: the three stages a program goes through are : development, use and maintenance

**Phases in the life cycle of software**

The software development process can be divided into 4 main phases: analysis, design, implementation and testing and debugging.

• Unified Modeling Language: the notation used to graphically depict a class and its members

**Object-Oriented Design ( OOD )**

• Learn about inheritance

• Learn about derived and base classes

• Explore how to redefine the member functions of a base class

• Examine how the constructors of base and derived classes work

• Learn how to construct the header file of a derived class

• Explore three types of inheritance: public, protected, and private

• Learn about composition

• Become familiar with the three basic principles of object-oriented design

• Learn about overloading

• Became aware of the restrictions on operator overloading

• Examine the pointer this

• Learn about friend functions

• Explore the members and nonmembers of a class

• Discover how to overload various operators

• Learn about templates

• Explore how to construct function templates and class templates

**A Survey of Object - Oriented Programming**

Here are a few important concepts, definitions and statements concerning the basics of object - oriented programming.

• The analysis of data and related functions into abstract classifications or classes is defined as **object - oriented** programming.

• A **class** encapsulates object - oriented data and functions. It defaults to a **private** access mode.

• A variable or property of a class is referred to as a **data member**.

• **Encapsulation** is the process of combining data members with the actions that are used to manipulate the type of data. This process is denoted with a class that contains both data and functions.

• Also referred to as an instance, an **object** is the actual variable associated with a class and its encapsulated properties and methods.

• Another term for an object, an **instance** is the actual variable that is associated with a class and its encapsulated properties and methods.

• The class **name** is used to declare objects of that class.

• A **constructor** is a specialized function that automatically executes when an object is created. The function’s name is the same as the class to which it belongs.

• Also referred to as polymorphism, **overloading** is the ability to have many versions of the same function and to select the appropriate version at run time.

• **public** and **private** are two access types of variables and functions.

• **Polymorphism** is the ability to have many versions of the same function and to select the appropriate version at run time.

• A **Hierarchy** is a relationship between classes. It promotes reusability of code and allows features to be added to a program without altering code that is already in use.

• **Inheritance** is a feature of object - oriented programming that allows one class to be derived from another class.

• A **base** class is the broadest type of class in a system that uses inheritance and object - oriented models.

• A **derived** class is the more specific type of a class in a hierarchy system.

• A derived class may override an **attribute** of a base class.

• Separating **interface** from implementation is a fundamental principle of good software engineering.

• The **implementation** of a class should be hidden from its clients.

• The goal of **information - hiding** is to make the object as robust and independent as possible. This assures that attribute values cannot be accidentally changed by other parts of the system.

**Exercises Involving Object - Oriented Programming**

**(1)** For each of the following, determine what data ( attributes ) may be of interest to someone considering purchasing the item:

(a) a college textbook

(b) a can of soda pop

(c) an automobile

**(2)** Paying attention to the important properties while ignoring inessential details is known as \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ .

(a) an abstraction

(b) selectiveness

(c) summarizing

(d) polymorphism

(e) functionality

Given the information below, identify each of the statements in Questions **(1)** through **(5)** as being true or false.

Geometrical figures called polygons are constructed only with straight line segments.  
Quadrilaterals are four - sided polygons that can be classified by either sides or angles. An important distinction involving quadrilaterals is whether one or more pairs of sides

are parallel. The most familiar quadrilaterals are parallelograms, and the most familiar

of these are rectangles and squares. Types of quadrilaterals are summarized below.

**Types of Quadrilaterals**

|  |  |  |
| --- | --- | --- |
| **Sample Figure** | **Description** |  |
|  |  |  |
| **Trapezoid** | A trapezoid is a quadrilateral with one pair of parallel sides. |  |
| **Parallelogram** | A parallelogram is a quadrilateral with two pairs of parallel sides. |  |
| **Rectangle** | A rectangle is a parallelogram with a right ( 90 degree ) angle  ( and consequently, four right angles ). |  |
| **Square** | A square is a rectangle with all sides having equal length. |  |
| **Rhombus** | A rhombus is a parallelogram with all sides having equal length. |  |

**(1)** All trapezoids are quadrilaterals. (a) True (b) False

**(2)** A square is both a rectangle and a quadrilateral. (a) True (b) False

**(3)** A trapezoid is a type of parallelogram. (a) True (b) False

**(4)** A rhombus is both a square and a parallelogram. (a) True (b) False

**(5)** Rectangles are a special type of parallelogram. (a) True (b) False

What is OOP?

Consider this acronym: P.I.E.

P Polymorphism

I Inheritance

E Encapsulation

Polymorphism

The ability to use the same expression and yet denote different operations.

Example: Please study your textbook an instructor could tell his/her students.

Students approach textbook studying in a different fashion. Some study immediate some wait for five minutes before class.

Inheritance

The ability to create new objects from existing objects.

I will use what YOU have the way I want to use it.

We utilize existing code to create new classes and objects.

Encapsulation

Combining data and operations on that data in a single unit.

Data would mean like payroll information and operations would be to calculate the gross pay of employees.

OOP involves working with classes and creating objects from the class that work with the class data members and methods.

When creating classes we often want to use that specialized function that is associated with each class and that is the class constructor.

What is a class constructor?

Here are some facts related to class constructors.

1. When an object is created, a class constructor is called.

**MyClass myObject;**

2. If a constructor is not coded in the class, the compiler provides you with a default.

3. A class constructor must have the same name as the class.

Example for the class named Newton, your constructor is

**public Newton() { }**

4. Class constructors do not have a return type, not even void.

5. A constructor is considered to be a specialized function for these reasons.

6. The main purpose of a constructor is to initialize class data members

( variables ) .

7. A constructor is generally declared as public. In this manner objects instantiated in the **main()** function can call this constructor.

8. Constructors can be overloaded. We can use the same function name but with a different argument list.

**Newton() { }**

**//generic constructor**

**Newton(int a, int b) { }**

**//overloaded constructor**

9. A constructor is generally called in the **main()** program method.

10. A class can have more than one constructor.