CANARA ENGINEERING COLLEGE

Object Oriented Programming with JAVA

Course Code: BCS306A

MODULE-1

CHAPTER-1

An Overview of Java

Instructor: Demian Antony DMello

Module Contents

Module-1

An Overview of Java: Object-Oriented Programming (Two Paradigms, Abstraction, The Three OOP Principles), Using Blocks of Code, Lexical Issues (Whitespace, Identifiers, Literals, Comments, Separators, The Java Keywords).

Data Types, Variables, and Arrays: The Primitive Types (Integers, Floating-Point Types, Characters, Booleans), Variables, Type Conversion and Casting, Automatic Type Promotion in Expressions, Arrays, Introducing Type Inference with Local Variables.

Operators: Arithmetic Operators, Relational Operators, Boolean Logical Operators, The Assignment Operator, The ? Operator, Operator Precedence, Using Parentheses.

Control Statements: Java's Selection Statements (if, The Traditional switch), Iteration Statements (while, do-while, for, The For-Each Version of the for Loop, Local Variable Type Inference in a for Loop, Nested Loops), Jump Statements (Using break, Using continue, return).

Textbook: Chapter 2, 3, 4, 5

Chapter Contents

An Overview of Java

Object-Oriented Programming (Two Paradigms, Abstraction, The Three OOP Principles), Using Blocks of Code, Lexical Issues (Whitespace, Identifiers, Literals, Comments, Separators, The Java Keywords).

Textbook: Chapter 2

Paradigms of program Construction

- Object-oriented programming (OOP) is at the core of Java.
- Computer programs consist of two elements: code and data.
- A program can be conceptually organized/constructed around its code or around its data.

Theoretical aspects of OOP: TWO Programming Paradigms

1. Process Oriented Model

- A program can be conceptually organized around its code.
- That is, programs are written around "what is happening"
- Characteristics: A program has a series of linear steps (that is, code) and the code acting on data.
- Procedural languages: Pascal, C and others

Paradigms of program Construction

2. Object Oriented Programming (OOP)

- A program can be conceptually organized around its data and a set of well-defined interfaces to that data.
- That is, programs are written around "who is being affected".
- Characteristics: Data controlling access to code.
- OOP languages: C++, C#, JAVA and others

Abstraction

- Managing complexity through abstraction
- **Example:** Car (many parts, used to drive) - Ignore the details of how the engine, braking etc. work. Instead, free to utilize the object as a whole.
- Hierarchical abstractions of complex systems can also be applied to traditional computer programs.
  1. The data from a traditional process-oriented program can be transformed by abstraction into its component objects.
  2. A sequence of process steps can become a collection of messages between these objects.
  3. Each object describes its own unique behavior and as a concrete entity, it respond to messages telling them to do something.

The Three OOP Principles

- Object-oriented programming languages provide mechanisms that help you implement the object-oriented model.
- They are Encapsulation, Inheritance, and Polymorphism.

Encapsulation (In Computing)

- Encapsulation is the mechanism that binds together code and the data it manipulates, and keeps both safe from outside interference and misuse.
- Analogously, encapsulation is as a protective wrapper that prevents the code and data from being arbitrarily accessed by other code defined outside the wrapper.
- Access to the code and data inside the wrapper is tightly controlled through a well-defined interface.

The Three OOP Principles

- **Example:** Automatic transmission on an automobile (Engine, gear shift, accelerator etc.).
- The power of encapsulated code is that every entity knows how to access it and thus can use it regardless of the implementation details—and without fear of unexpected side effects.

Encapsulation (In Java)

- In Java, the basis of encapsulation is the class. A class defines the structure and behavior (data and code) that will be shared by a set of objects.
- Each object of a given class contains the structure and behavior defined by the class.

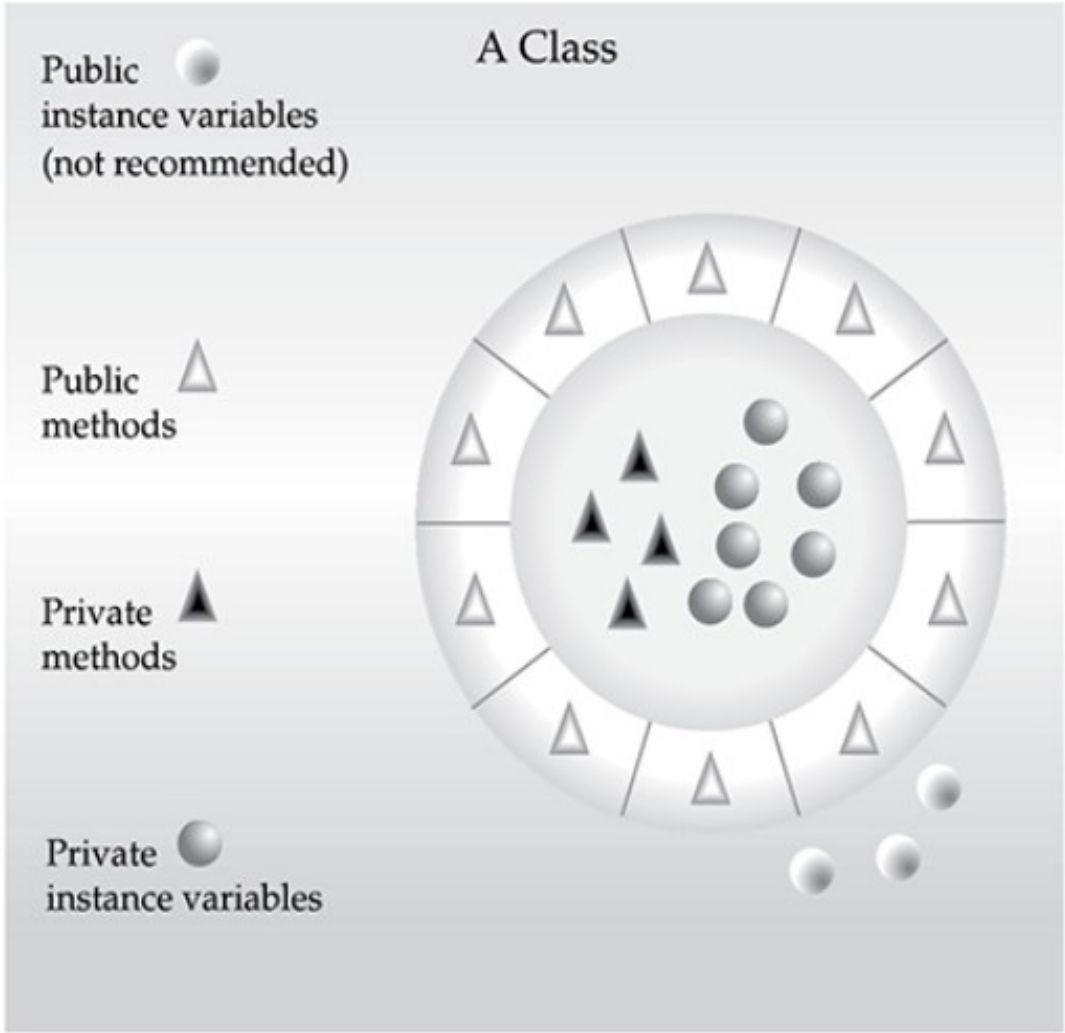
The Three OOP Principles

- For this reason, objects are sometimes referred to as instances of a class. Thus, a class is a logical construct; an object has physical reality.
- **Class Creation/Definition:** When you create a class, you will specify the code and data that constitute that class. Collectively, these elements are called members of the class. Specifically, the data defined by the class are referred to as member variables or instance variables. The code that operates on that data is referred to as member methods or just methods.
- In properly written Java programs, the methods define how the member variables can be used/accessed.
- Information Hiding: There are mechanisms for hiding the complexity of the implementation inside the class.

The Three OOP Principles

- Each method or variable in a class may be marked private or public. These are termed as visibility markers.
- The public interface (methods) of a class represents everything that external users of the class need to know, or may know.
- The private methods and data can only be accessed by code that is a member of the class. Therefore, any other code that is not a member of the class cannot access a private method or variable.
- The private members of a class may only be accessed by other parts of your program through the class' public methods, you can ensure that no improper actions take place (Accessors and Mutators).

The Three OOP Principles



Visibility Markers

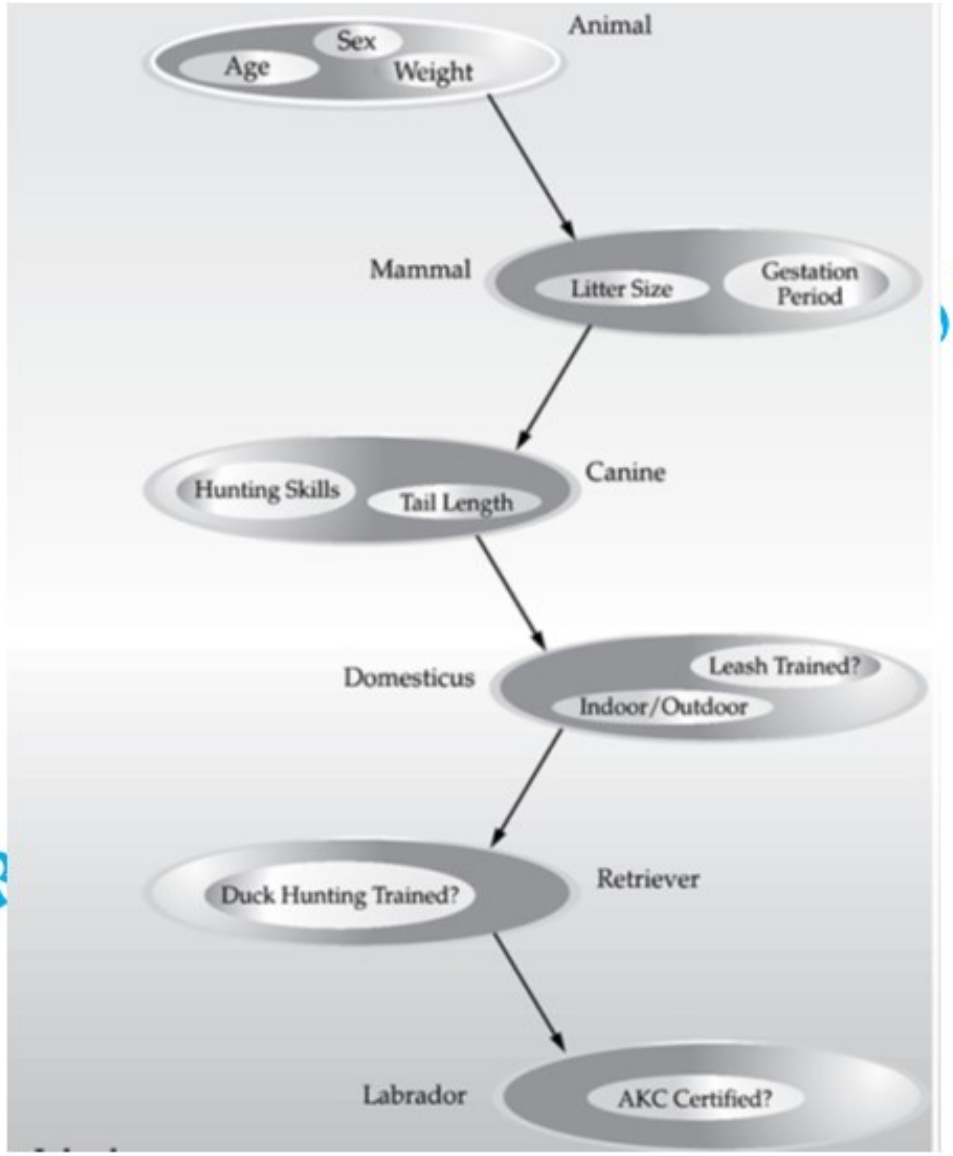
The Three OOP Principles

Inheritance

- Inheritance is the process by which one object acquires the properties of another object.
- Inheritance supports the concept of hierarchical classification and most knowledge is made manageable by hierarchical (that is, top-down) classifications.
- Example: A Golden Retriever is part of the classification dog, which in turn is part of the mammal class, which is under the larger class animal.
- Without the use of hierarchies, each object would need to define all of its characteristics explicitly.

The Three OOP Principles

Example:
Inheritance



The Three OOP Principles

- By use of inheritance, an object defines only those qualities that make it unique within its class. It can inherit its general attributes from its parent. Thus, it is the inheritance mechanism that makes it possible for one object to be a specific instance of a more general case.
- **Example:** If you wanted to describe a more specific class of animals, such as mammals, they would have more specific attributes, such as type of teeth and mammary glands. This is known as a subclass of animals, where animals are referred to as mammals' superclass.
- Since mammals are simply more precisely specified animals, they inherit all of the attributes from animals. A deeply inherited subclass inherits all of the attributes from each of its ancestors in the class hierarchy.

The Three OOP Principles

- Inheritance interacts with encapsulation as well. If a given class encapsulates some attributes, then any subclass will have the same attributes plus any that it adds as part of its specialization.
- This is a key concept that lets object-oriented programs grow in complexity linearly rather than geometrically.
- A new subclass inherits all of the attributes of all of its ancestors. It does not have unpredictable interactions with the majority of the rest of the code in the system.

The Three OOP Principles

Polymorphism

- Polymorphism (from Greek, meaning "many forms") is a feature that allows one interface to be used for a general class of actions. The specific action is determined by the exact nature of the situation.
- More generally, the concept of polymorphism is often expressed by the phrase "one interface, multiple methods." This means that it is possible to design a generic interface to a group of related activities.
- This helps reduce complexity by allowing the same interface to be used to specify a general class of action.
- It is the compiler's job to select the specific action (that is, method) as it applies to each situation.

The Three OOP Principles

- When properly applied, polymorphism, encapsulation, and inheritance combine to produce a programming environment that supports the development of far more robust and scaleable programs than does the process-oriented model.
- A well-designed hierarchy of classes is the basis for reusing the code in which you have invested time and effort developing and testing.
- Encapsulation allows you to migrate your implementations over time without breaking the code that depends on the public interface of your classes.
- Polymorphism allows you to create clean, sensible, readable, and resilient code.

Using Blocks of Code

- Java allows two or more statements to be grouped into blocks of code, also called code blocks. This is done by enclosing the statements between opening and closing curly braces.
- Once a block of code has been created, it becomes a logical unit that can be used any place that with a single or multiple statements.
- Example:

```
if(x < y) { // begin a block
  x = y;
  y = 0;
} // end of block
```

```
class BlockTest {                                   Bloack-3
  public static void main(String[] args) {
    int x, y;                                       Bloack-2

    y = 20;

    // the target of this loop is a block
    for(x = 0; x<10; x++) {                         Bloack-1
      System.out.println("This is x: " + x);
      System.out.println("This is y: " + y);
      y = y - 2;
    }
  }
}
```

Lexical Issues

- Java programs are a collection of whitespace, identifiers, literals, comments, operators, separators, and keywords.

Whitespace

- Java is a free-form language. This means that you do not need to follow any special indentation rules.
- For instance, the program can be written all on one line or any other way, as long as there was at least one whitespace character between each token that was not already delineated by an operator or separator.
- In Java, whitespace includes a space, tab, newline, or form feed.

Lexical Issues

Identifiers

- Identifiers are used to name program elements, such as classes, variables, and methods.
- An identifier may be any descriptive sequence of uppercase and lowercase letters, numbers, or the underscore and dollar-sign characters. (The dollar sign character is not intended for general use.)
- They must not begin with a number, lest they be confused with a numeric literal.
- Valid Identifiers:

AvgTemp	count	a4	$test	this_is_ok

Lexical Issues

Literals

- A constant value in Java is created by using a *literal* representation of it.

100	98.6	'X'	"This is a test"

- For example: Left to right, the first literal specifies an integer, the next is a floating-point value, the third is a character constant, and the last is a string.
- A literal can be used anywhere a value of its type is allowed.

Comments

- There here are three types of comments defined by Java: single-line, multiline and a documentation comment.
- A single-line comment begins with a // and ends at the end of the line.

```
//Your program begins with a call to main().
```

Lexical Issues

- Multiline comment must begin with /* and end with */. Anything between these two comment symbols is ignored by the compiler.
- As the name suggests, a multiline comment may be several lines long.

```
/*
   This is a simple Java program.
   Call this file "Example.java".
*/
```

- Documentation comment is used to produce an HTML file that documents your program. The documentation comment begins with a /** and ends with a */.

Separators

- In Java, there are a few characters that are used as separators. The most commonly used separator in Java is the semicolon. Semicolon is often used to terminate statements.

Lexical Issues

Symbol	Name	Purpose
( )	Parentheses	Used to contain lists of parameters in method definition and invocation. Also used for defining precedence in expressions, containing expressions in control statements, and surrounding cast types.
{ }	Braces	Used to contain the values of automatically initialized arrays. Also used to define a block of code, for classes, methods, and local scopes.
[ ]	Brackets	Used to declare array types. Also used when dereferencing array values.
;	Semicolon	Terminates statements.
,	Comma	Separates consecutive identifiers in a variable declaration. Also used to chain statements together inside a **for** statement.
.	Period	Used to separate package names from subpackages and classes. Also used to separate a variable or method from a reference variable.
::	Colons	Used to create a method or constructor reference.
...	Ellipsis	Indicates a variable-arity parameter.
@	At-sign	Begins an annotation.

Separators

The Java Keywords

- There are 67 keywords currently defined in the Java language.

Lexical Issues

- These keywords, combined with the syntax of the operators and separators, form the foundation of the Java language.
- In general, keywords cannot be used as identifiers, meaning that they cannot be used as names for a variable, class, or method.
- In addition to the keywords, Java reserves three other names that have been part of Java from the start: true, false, and null. These are values defined by Java.
- You may not use these words for the names of variables, classes, and so on.

The Java Class Libraries

- The Java environment relies on several built-in class libraries that contain many built-in methods that provide support for such things as I/O, string handling, networking, and graphics.

Lexical Issues

- The standard classes also provide support for a graphical user interface (GUI). Thus, Java as a totality is a combination of the Java language itself, plus its standard classes.

JAVA Keywords

abstract	assert	boolean	break	byte	case
catch	char	class	const	continue	default
do	double	else	enum	exports	extends
final	finally	float	for	goto	if
implements	import	instanceof	int	interface	long
module	native	new	non-sealed	open	opens
package	permits	private	protected	provides	public
record	requires	return	sealed	short	static
strictfp	super	switch	synchronized	this	throw
throws	to	transient	transitive	try	uses
var	void	volatile	while	with	yield
_					

Java Program – Few Instructions

```
class Example {
  // Your program begins with a call to main().
  public static void main(String[] args) {
    System.out.println("This is a simple Java program.");
  }
}
```

- In Java, all code must reside inside a class. By convention, the name of the main class should match the name of the file that holds the program.
- For the above example, the source file should be **Example.java**.
- Java is case-sensitive.
- To compile the Example program, execute the compiler, javac, specifying the name of the source file on the command line, as shown here:

  C:\>javac Example.java

- The javac compiler creates a file called **Example.class** that contains the bytecode version of the program.

Java Program – Few Instructions

- To actually run the program, you must use the Java application launcher called java. To do so, pass the class name Example as a command-line argument, as shown here:
- C:\>java Example
- As a general rule, a Java program begins execution by calling main( ).
- The main( ) must be declared as public, since it must be called by code outside of its class when the program is started.
- The keyword static allows main( ) to be called without having to instantiate a particular instance of the class. This is necessary since main( ) is called by the Java Virtual Machine before any objects are made.
- The keyword void simply tells the compiler that main( ) does not return a value.
- Remember - main( ) is the method called when a Java application begins.

Review Questions

1. Explain how Object Oriented Programming model is different from Process/procedure driven programming model?
2. Define the term 'Abstraction' and how 'abstraction' applied to traditional computer programs?
3. Explain three OOP principles: Encapsulation, Inheritance and Polymorphism.
4. Explain the following with a suitable JAVA program segment.
   a. Code blocks
   b. Comments
   c. Identifiers
   d. Literals