Final Report

Explanations

Instance Variable in Java

Instance variable in Java is used by Objects to store their states. Variables which are defined without the STATIC keyword and are Outside any method declaration are Object-specific and are known as instance variables. They are called so because their values are instance specific and are not shared among instances.

Naming Rules for Variables

The names of variables in the Java language are referred to as identifiers. The best naming convention is to choose a variable name that will tell the reader of the program what the variable represents. Variable names that describe the data stored at that particular memory location are called mnemonic variables.

- All variable names must begin with a letter of the alphabet, an underscore, or (_), or a dollar sign ($). The convention is to always use a letter of the alphabet. The dollar sign and the underscore are discouraged.
- After the first initial letter, variable names may also contain letters and the digits 0 to 9. No spaces or special characters are allowed.
- The name can be of any length, but don't get carried away. Remember that you will have to type this name.
- Uppercase characters are distinct from lowercase characters. Using ALL uppercase letters are primarily used to identify constant variables. Remember that variable names are case-sensitive.
- You cannot use a java keyword (reserved word) for a variable name.

Naming Conventions

Naming conventions make programs more understandable by making them easier to read. They can also give information about the function of the identifier-for example, whether it's a constant, package, or class-which can be helpful in understanding the code.

Identifier Type	Rules for Naming	Examples
Packages	The prefix of a unique package name is always written in all-lowercase ASCII letters and should be one of the top-level domain names, currently com, edu, gov, mil, net, org, or one of the English two-letter codes identifying countries as specified in ISO Standard 3166, 1981. Subsequent components of the package name vary according to an organization's own internal naming conventions. Such conventions might specify that certain directory name components be division, department, project, machine, or login names.	com.sun.eng com.apple.quicktime.v2 edu.cmu.cs.bovik.cheese
Classes	Class names should be nouns, in mixed case with the first letter of each internal word capitalized. Try to keep your class names simple and descriptive. Use whole words-avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML).	class Raster; class ImageSprite;
Interfaces	Interface names should be capitalized like class names.	interface RasterDelegate; interface Storing;
Methods	Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal	run(); runFast();

	word capitalized.	getBackground();
Variables	Except for variables, all instance, class, and class constants are in mixed case with a lowercase first letter. Internal words start with capital letters. Variable names should not start with underscore _ or dollar sign $ characters, even though both are allowed. Variable names should be short yet meaningful. The choice of a variable name should be mnemonic- that is, designed to indicate to the casual observer the intent of its use. One-character variable names should be avoided except for temporary "throwaway" variables. Common names for temporary variables are i, j, k, m, and n for integers; c, d, and e for characters.	int i; char c; float myWidth;

Constants	The names of variables declared class constants and of ANSI constants should be all uppercase with words separated by underscores ("_"). (ANSI constants should be avoided, for ease of debugging.)	static final int MIN_WIDTH = 4;
		static final int MAX_WIDTH = 999;
		static final int GET_THE_CPU = 1;

Wrapper Classes in Java

A Wrapper class is a class whose object wraps or contains a primitive data types. When we create an object to a wrapper class, it contains a field and in this field, we can store a primitive data types. In other words, we can wrap a primitive value into a wrapper class object.

Need of Wrapper Classes

1. They convert primitive data types into objects. Objects are needed if we wish to modify the arguments passed into a method (because primitive types are passed by value).
2. The classes in java.util package handles only objects and hence wrapper classes help in this case also.
3. Data structures in the Collection framework, such as ArrayList and Vector, store only objects (reference types) and not primitive types.
4. An object is needed to support synchronization in multithreading.

Primitive Data types and their Corresponding Wrapper Classes

Primitive Data Type	Wrapper Class
char	Character
byte	Byte
short	Short
long	Integer
float	Float
double	Double
boolean	Boolean

Autoboxing and Unboxing

Autoboxing: Automatic conversion of primitive types to the object of their corresponding wrapper classes is known as autoboxing. For example – conversion of int to Integer, long to Long, double to Double etc.

Unboxing: It is just the reverse process of autoboxing. Automatically converting an object of a wrapper class to its corresponding primitive type is known as unboxing. For example – conversion of Integer to int, Long to long, Double to double etc.

Constructors in Java

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory. It is a special type of method which is used to initialize the object. Every time an object is created using the new() keyword, at least one constructor is called. It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

There are two types of constructors in Java: no-arg constructor, and parameterized constructor.
Note: It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

Rules for creating Java constructor

1) There are two rules defined for the constructor.
2) Constructor name must be the same as its class name
3) A Constructor must have no explicit return type
4) A Java constructor cannot be abstract, static, final, and synchronized

Primitive Data types in Java

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Store fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

Initializer in Java

An initializer is a line of code (or a block of code) placed outside any method, constructor, or other block of code. Initializers are executed whenever an instance of a class is created, regardless of which constructor is used to create the instance.

Java Object finalize() Method

Finalize() is the method of Object class. This method is called just before an object is garbage collected. finalize() method overrides to dispose system resources, perform clean-up activities and minimize memory leaks.

Syntax

protected void finalize() throws Throwable

Garbage Collector

Garbage Collector is a program that manages memory automatically wherein de-allocation of objects is handled by Java rather than the programmer. In the Java programming language, dynamic allocation of objects is achieved using the **new** operator. An object once created uses some memory and the memory remains allocated till there are references for the use of the object. When there are no references to an object, it is assumed to be no longer needed, and the memory, occupied by the object can be reclaimed. There is no explicit need to destroy an object as Java handles the de-allocation automatically. The technique that accomplishes this is known as **Garbage Collection**. Programs that do not de-allocate memory can eventually crash when there is no memory left in the system to allocate. These programs are said to have memory leaks.
Garbage collection in Java happens automatically during the lifetime of the program, eliminating the need to de-allocate memory and thereby avoiding memory leaks. In C language, it is the programmer's responsibility to de-allocate memory allocated dynamically using free() function. This is where Java memory management leads. Note: All objects are created in **Heap** Section of memory.

The Class Declaration

The class declaration component declares the name of the class along with other attributes such as the class's superclass, and whether the class is public, final, or abstract. At minimum, the class declaration must contain the class keyword and the name of the class that you are defining. Thus the simplest class declaration that you can write looks like this:

class *NameOfClass* {
. . .

}

For example, this code snippet declares a new class named ImaginaryNumber.

```
class ImaginaryNumber {
   . . .
}
```

Class names must be a legal Java identifier and, by convention, begin with a capital letter. Often, a minimal class declaration such as this one is all you'll need. However, the class declaration can say more about the class. More specifically, within the class declaration you can:

- declare what the class's superclass is
- list the interfaces implemented by the class
- declare whether the class is abstract, final or public