INTRODUCTION

- JAVA(Part 2)

Prepared by Renetha J.B.(LMCST)

Topics

- ✓ Java applet
- ✓ Java Buzzwords
- ✓ Java program structure
- ✓ Comments
- ✓ Garbage Collection
- ✓ Lexical Issues

Java applet

- An applet is a special kind of **small Java program** that is designed to be
  - **transmitted over the Internet** and
  - **automatically executed by a Java-compatible web browser.**
- If the user clicks a link that contains an applet, the applet will be automatically downloaded and run in the browser.
- The **applets** are **used** to provide interactive features to web applications.

Java applet(contd.)

- Applets are typically used to
  - **display data** provided by the server,
  - **handle user input**, or
  - provide **simple functions**, such as a loan calculator, that **execute locally**, rather than on the server.
- Applet allows some functionality to be moved from the server to the client.
- Applet is a **dynamic, self-executing program.**

Applet - features

- Usually a program that downloads and executes automatically on the client computer must be prevented from doing harm. It must also be able to run in a variety of different environments.
- Java solved these problems in an effective and elegant way.
  - **Security-** Java achieved security by confining an applet to the Java execution environment and **not allowing it access to other parts of the computer.**
  - **Portability-** Same applet can be downloaded and executed by the wide variety of CPUs, operating systems, and browsers connected to the Internet.

Applet(contd.)

- Applets are
  - small applications
  - that are accessed on an Internet server,
  - transported over the Internet,
  - automatically installed,
  - and run as part of a web-document.

Applet(contd

- Applets are **not stand-alone programs**.
  - Instead, they run within either a web browser or an applet viewer.
- JDK provides a standard applet viewer tool called **applet viewer**.
- In general, **execution of an applet does not begin at main() method.**

Lifecycle of Java Applet

Java Applet vs Java Application

Java Application	Java Applet
Java Applications are the stand-alone programs which can be executed independently	Java Applets are small Java programs which are designed to exist within HTML web document
Java Applications must have main() method for them to execute	Java Applets do not need main() for execution
Java Applications Just needs the JRE	Java Applets cannot run independently and require API's
Java Applications do not need to extend any class unless required	Java Applets must extend java.applet.Applet class
Java Applications can execute codes from the local system	Java Applets Applications cannot do so
Java Applications has access to all the resources available in your system	Java Applets has access only to the browser-specific services

Java Buzzwords

- Simple
- Secure
- Portable
- Object-oriented
- Robust
- Multithreaded
- Architecture-neutral
- Interpreted
- High performance
- Distributed
- Dynamic

Java Buzzwords(contd.)

- **Simple**
  - easy for the professional programmer to learn and use effectively.
- **Security**
  - Java achieved security by confining to the Java execution environment and not allowing it access to other parts of the computer.
- **Portable**
  - Same Java program can be executed by the wide variety of CPUs, operating systems etc.

Java Buzzwords(contd.)

- **Object-Oriented**
  - Atleast one class should be there.
  - everything is an object.
  - The object model in Java is simple and easy to extend, while primitive types, such as integers, are kept as high-performance nonobjects

Java Buzzwords(contd.)

- **Robust**
  - the program **must execute reliably in a variety of system.**
  - Ability to create robust programs was given a high priority in the design of Java.
- To gain reliability,
  - Java restricts us in a few key areas to **force us to find mistakes** early in program development.
  - Java **frees us** from having to worry about many of the **most common causes of programming errors**.
  - Because Java is a strictly typed language, it **checks code at compile time**. However, it also **checks code at run time**.

Why Java is Robust?(contd.)

- Two main reasons for program failure are **memory management mistakes** and **mishandled exceptional conditions** (that is, run-time errors).
  - Memory management and Exception can be a difficult, tedious task in traditional programming environments.
- Java virtually eliminates these problems by **managing memory allocation and deallocation** for us.
- Java helps in handling exceptional conditions by **providing object-oriented exception handling.**
- **So Java is robust.**

Java Buzzwords(contd.)

- **Multithreaded**
  - Java was designed to meet the real-world requirement of creating interactive, networked programs.
  - To accomplish this, Java supports multithreaded programming, which allows to **write programs that do many things simultaneously.**
- **Architecture-Neutral**
  - A central issue for the Java designers was that of code longevity and portability.
  - Their goal was "write once; run anywhere, any time, forever." **WORA**

Java Buzzwords(contd.)

- **Interpreted and High Performance**
  - Java enables the creation of cross-platform programs by compiling into an intermediate representation called Java bytecode, which can be executed by JVM.
  - Java bytecode can be easily **translated directly into native machine code** for **very high performance** **by using a just-in-time compiler.(JIT)**
  - Java run-time systems that provide this feature.

Java Buzzwords(contd.)

- **Distributed**
  - Java is designed for the distributed environment of the Internet because **it handles TCP/IP protocols.**
  - Java also supports ***Remote Method Invocation (RMI).***
    - *This feature enables a program to* invoke methods across a network.

Java Buzzwords(contd.)

- **Dynamic**
  - Java programs **carry** with them substantial amounts of **run-time type information** that is used to verify and resolve accesses to objects at run time.
  - This makes it possible to **dynamically link code** in a safe and expedient manner.
  - Small fragments of bytecode may be dynamically updated on a running system.

Java

- Java is related to C++, which is a direct descendant of C.
- From C, Java derives its syntax.
- Many of Java's object-oriented features were influenced by C++.

Java program

- Write and save the source file(program code) as **Example.java** in any directory.

  E.g. D:\oop

- **Compile** the program

  javac Example.java

  – If there is ***error,*** then correct and save and compile program again.

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

Java program

- If there is **no error after compilation** you can **execute** the program using Java application launcher, called **java**

  java Example

  Here we are passing the class name *Example* as a command-line argument**.**
- i.e. When we execute **java command** we are actually specifying the name of the class that you want to execute.
  - It will automatically search for a file by that **name that has the .class extension.** ***(Here Example.class)***
  - **If it finds the file, it will execute the code** contained in the specified class.

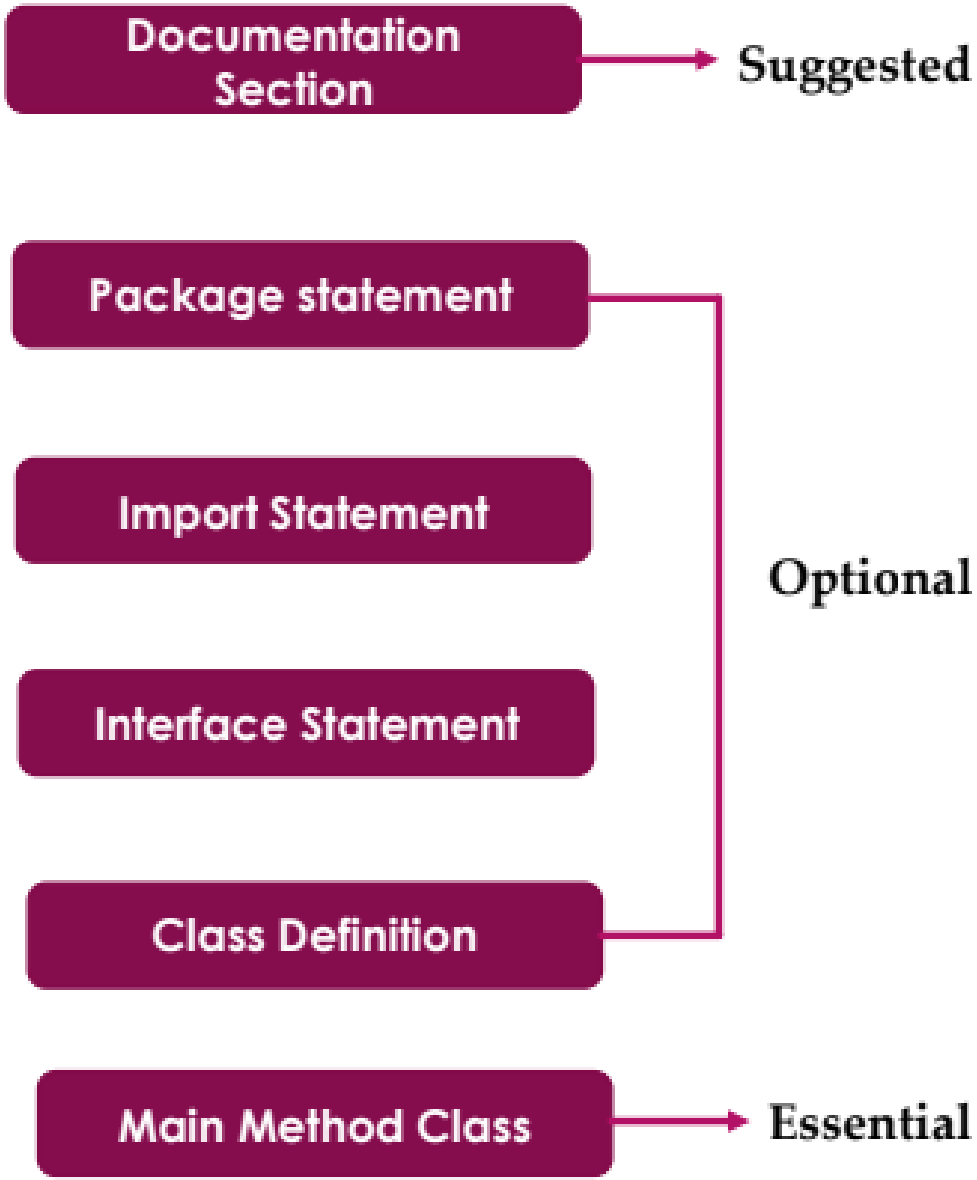
Java program

- In Java, a source file is officially called a *compilation unit*.
- *It is a file that contains one* or more class definitions.
- The Java compiler requires that a source file use the **.java** filename extension.
- In Java, **all code must reside inside a class.**
- By convention, **the name of that class in program should match the name of the file** that holds the program.
- Java programs are **case sensitive.**

Java program

- If there is **no public class** then **file name** may be same or different from the **class name** in **Java**.
- If there is a **public class** then the file name is same as the public class.

Java program structure



Documentation Section

- We can write comments in this section.
- Comments are beneficial for the programmer because they help them understand the code.
- There are three types of comments that Java supports
  - Single line Comment
  - Multi-line Comment
  - Documentation Comment

Comments

- Single line Comment
  - To comment a single line
  - Line start with //
- Multi-line Comment
  - To comment many lines
  - Start with /* and end with */
- Documentation Comment
  - Multi-line comment .
  - A **doc comment** appears immediately before a class, interface, method, or field definition and contains **documentation** for that class, interface, method, or field.
  - To document the API of the code.
  - Start with /* and end with */

Comments

- Single line comment

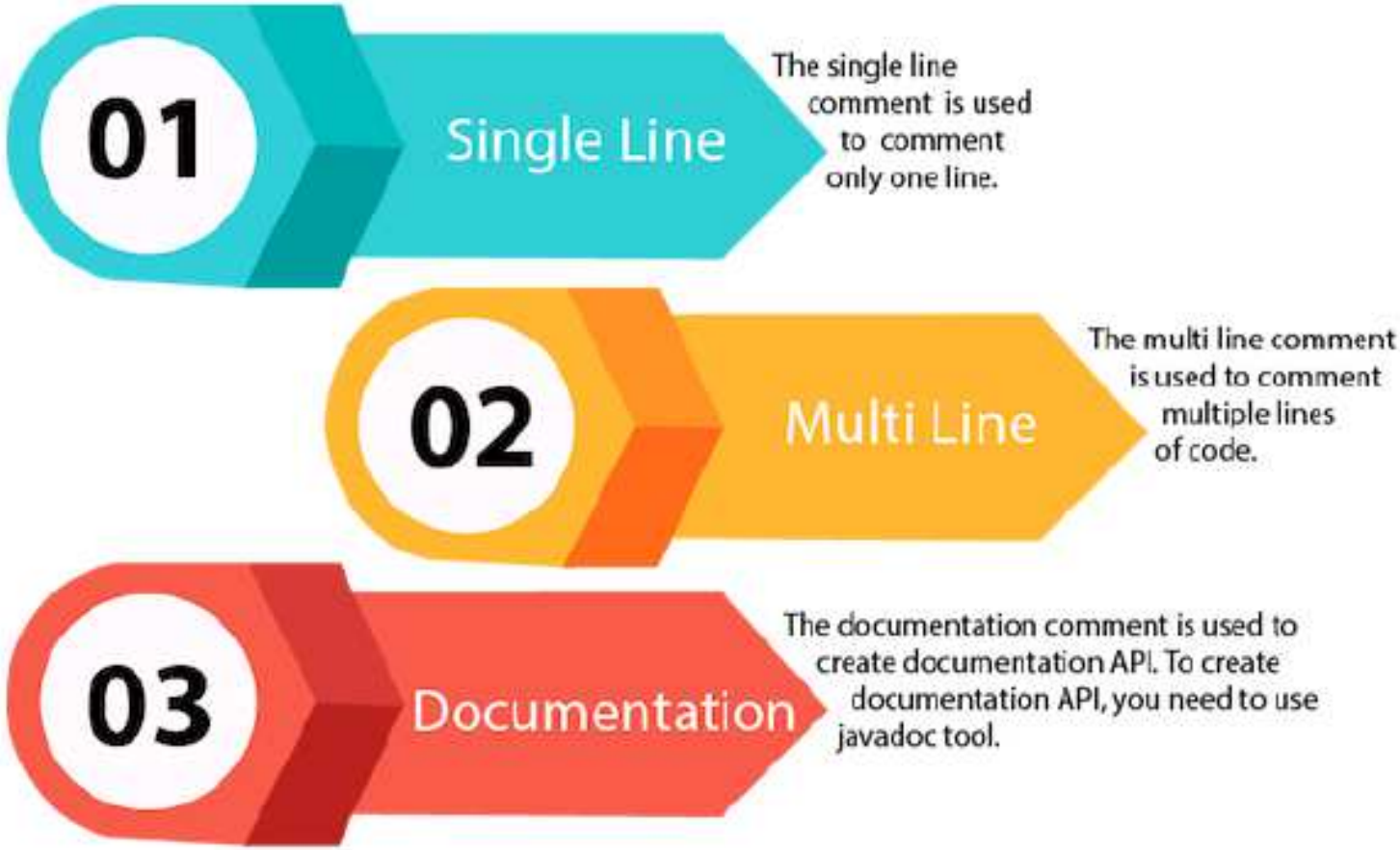
  //This is single line comment
- Multi-line Comment

  /* this is multiline comment.

  and support multiple lines*/
- Documentation Comment

```
/**
*this is documentation
 *comment
*/
```

Comments


01
Single Line
The single line comment is used to comment only one line.
//
02
Multi Line
The multi line comment is used to comment multiple lines of code.
/*
*/
03
Documentation
The documentation comment is used to create documentation API. To create documentation API, you need to use javadoc tool.
/** */

Package Statement

- A package is **a group of classes that are defined by a name.**
  - That is, if you want to ***declare many classes within one element***, then you can declare it within a package
- If you do not want to declare any package, then there will be no problem with it, and you will not get any errors.(Package is optional)
- Package is declared as:

package package_name;

Eg: package mypackage;

We can create a package with any name.

Import Statement

- If we want to **use a class of another package**, then we have to import it directly into your program using **import statement.**
- Once imported, a class can be referred to directly, using only its name.
- **Syntax**

import packagename;

Import Statement(contd.)

- Many predefined classes are stored in packages in Java.
- We can import a specific class or classes in an import statement.
- Examples:

import java.io.*; *// import **all** classes from java.io package*

import java.util.Date; //imports the Date class

import java.applet.*; /*imports **all** the classes from the *java.applet* package*/

Interface Statement

- This statement is used to specify interface in Java.
- Interfaces are like a class that includes a group of method declarations.
- It's an optional section and can be used when programmers want to implement multiple inheritances within a program.

Class Definition

- A Java program may contain several class definitions.
- Classes are the **main and essential elements** of any Java program.
- A class is a collection of data and methods.
- It is a good convention to start class name with capital letter
- E.g

```
class Example{
}
```

Main Method Class

- Every Java stand-alone program requires the main method as the starting point of the program.
  - This is an essential part of a Java program.

- There may be many classes in a Java program, and **only one class defines the main method**.
- Methods contain data type declaration and executable statements.

```
class Classname
{
    public static void main(String args[])
    {

    }
    }
}
```

Main function

```
public static void main(String args[])
{


}
```

- **public** means it **can also be used by code outside of its class.**
- **static** is used when **we want to access a method without creating its object.**
- **void** indicates that a **function does not return a value**.
- **main is the function name**
- **String is a class.**
  - **args is an array of instances of String**

public static void main

- When the main method is declared **public**, it means that it **can also be used by code outside of its class**.
- The word ***static*** is used when ***we want to access a method without creating its object*** (because we call the main method, before creating any class objects*)*
- The word **void** indicates that a **function does not return a value**.
  - main( ) does not return a value.
  - *main() is the starting point of a Java program.*
- Proper spaces can be given in program to make program *easier to read and understand* -It is called *indentation.*

String args[ ]

- **String args[ ]** is an array of instances of class String.
  - Each element is a string, which has been named as "args".
- *We can use any name instead of args.*
- If we Java program is run through the console, we can pass the input parameters (command line arguments), to main( ) method.
  - args **receives any command-line arguments present when the program is executed.**
- **String args[ ]** can also be written as **String[] args**

System.out.println();

- System.out.println();
- This statement is used to **print a text on the screen** as output .
- **System** is a ***predefined class.***
- **out** is an ***object of the PrintWriter class*** defined in the system
- The method **println()** *prints the text that is inside the ( ) on the screen and add a new line.*
- We can also use **print**() method instead of println() method.(new line will not be added)
- All Java statement ends with a semicolon.

Simple Java program

FIRST SIMPLE PROGRAM

```
/*
This is a simple Java program.
 Save this file "Example.java".
*/
import java.io.*;
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.");
      }
}
```

- This program begins with the following lines:

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

- *This is a multiline comment.- more than one lines are commented. Lines inside /* and */ are comments*
  - *NOT EXECUTABLE CODES*
- The contents of a comment are **ignored** by the compiler. Instead, a comment describes or explains the operation of the program to anyone who is reading its source code.

- The next line of code in the program is :

  import java.io.*;

This statement imports all classes in the java.io package.

During compilation , the compiler will look at those classes.

- java.io.* is **Input/Output package**

- The next line of code in the program is :

class Example

{

- This line uses the keyword **class** to declare that a new class is being defined.
  - *Here Example is the* identifier that is the name of the class.
- The entire class definition, including all of its members, will be between the opening curly brace ({) and the closing curly brace (}).

- The next line in the program is the ***single-line comment,***

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

- Next line

public static void main(String args[])

This is the main function header

- Main function definition

```
public static void main(String args[])
{
   System.out.println("This is a simple Java program.");
}
```

- **System.out.println**("This is a simple Java program.");
  - prints the line **This is a simple Java program.** as output on the screen and control goes to new line

Simple Java program


```
FIRST SIMPLE PROGRAM
/*
This is a simple Java program.
 Save this file "Example.java".
*/
import java.io.*
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.");
      }
}
```

D:/OOP> javac Example.java

D:/OOP> java Example

This is a simple Java program.

Garbage Collection in Java

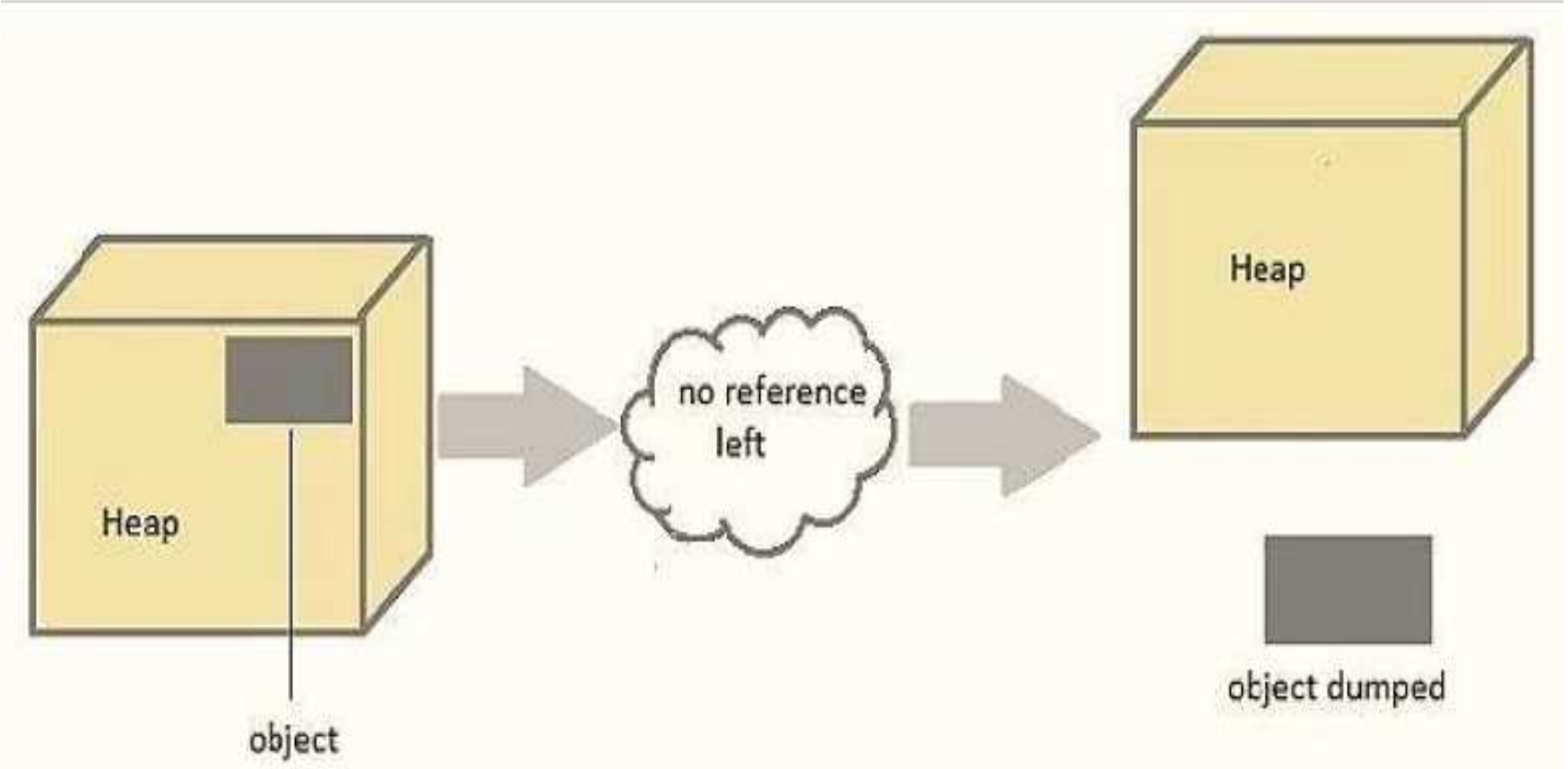
- Garbage collection is a process of **releasing unused memory.**
  - objects can be **dynamically allocated using the new operator** which can be **automatically deallocated using garbage collector.**
- When **no references to an object** exist, that **object** is assumed to be no longer needed, and the memory occupied by the object can be reclaimed.
  - The technique that accomplishes this is called *garbage collection.*

Garbage Collection in Java

- When JVM starts up, it creates a **heap area** which is known as runtime data area. This is where all the objects (instances of class) are stored.
- Since this heap area is limited, it is required to ***manage this area efficiently*** by **removing the objects that are no longer in use**.

Garbage collection(contd.)

- The process of r**emoving unused objects from heap memory** is known as **Garbage collection**
  - this is a part of ***memory management*** in Java.
- Languages like C/C++ don't support automatic garbage collection,
- In Java, the **garbage collection is automatic.**


Heap
object
no reference
left
Heap
object dumped

Garbage collection(contd.)

- In java, **garbage** means ***unreferenced objects***.
- Main objective of Garbage Collector is to free heap memory by destroying unreachable objects.
- **Unreachable objects :** An object is said to be unreachable if and only if(iff) it **doesn't contain any reference to it**.
- Eligibility for garbage collection : An object is said to be eligible for GC(garbage collection) iff it is unreachable.

Request for Garbage Collection

- We can request to JVM for garbage collection however, it is upto the JVM when to start the garbage collector.
- Java **gc() method** is used to **call garbage collector explicitly**.
  - However gc() method does not guarantee that JVM will perform the garbage collection.
  - It only **request the JVM for garbage collection**. This method is present in System and Runtime class.

finalize() method

- **finalize**() method – This method is **called** each time just **before the object is garbage collected** and it **perform cleanup processing.**
  - By using finalization, we can **define specific actions** that will occur when an object is just about to be reclaimed by the garbage collector.
  - E.g if an object is holding some non-Java resource such as a file handle or character font, then we might want to make sure these resources are freed before an object is destroyed.

Garbage collection(contd.)

- The Garbage collector of JVM collects **only those objects that are created by new keyword.**
- So if we have created any object **without new**, we can use **finalize() method** to **perform cleanup processing**

Lexical Issues

- Java programs are a collection of JAVA **TOKENS**
  - Whitespace
  - Identifiers
  - Literals
  - Comments
  - Operators
  - Separators
  - Keywords.

Lexical Issues- Whitespace

- Java is a **free-form language**.
  - We ***do not need to follow any special indentation rules***.
  - It means that we can write statements(code) in java program **all on one line** or **in any other strange way.**
- There should be **at least one whitespace character between each token** (if it is not already delineated by an operator or separator).
- In Java, **whitespace** is a
  - **Space**
  - **Tab**
  - **Newline.**

Lexical Issues-Identifiers

- Identifiers are used for class names, method names, and variable names.
- An identifier may be any descriptive sequence of
  - **Uppercase letters ABCD……Z**
  - **Lowercase letters abcd…z**
  - **Numbers 0123456789**
  - **Underscore _**
  - **Dollar-sign characters $**
- They must not begin with a number.
- Java is case-sensitive

Lexical Issues-Identifiers

• AvgTemp	• Valid
• count	• Valid
• 2count	• Invalid. should not begin with number
• a4	• Valid
• Not/ok	• Invalid. / not allowed
• $test	• Valid
• this_is_ok	• Valid
• high-temp	• Invalid - not allowed

Lexical Issues- Comments

- **Single line**
- **Multi-line**
- **Documentation comment**

Lexical Issues- Literals

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

  100 integer

  98.6 floating point

  'X' character

  "This is a test" string
- A literal can be used anywhere a value of its typeis allowed.

Operators

- Operators are used for performing operations.
  - **Arithmetic Operators**
    - **+, -, *, /, %**
    - **++, +=, -=,*=, /=, %=, - -**
  - **Bitwise Operators** ~ Bitwise unary NOT
    - &,| ,^
    - >> ,>>> ,<<
    - &=, |=, ^=, >>=, >>>=
  - **Relational Operators**
    - = =, !=, >, < , <=, >=
  - **Logical Operators**
    - **&, |, ^, ||,&&,** &=,| =, ^=, ==, !=, ?:

Lexical Issues- Separators

- In Java, there are a few characters that are used as separators.
- Separator is a symbols that is used to **separate a group of code from one another.**
- The most commonly used separator in Java is the **semicolon**. **;**

Lexical Issues- Separators(contd.)

The separators are

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.

Lexical Issues- The Java Keywords

- Keywords are **reserved words.**
- There are 50 keywords currently defined in the Java language.
- These keywords, combined with the syntax of the operators and separators, form the foundation of the Java language.
- These **keywords cannot be used as names for a variable, class, or method.**

Lexical Issues- The Java Keywords

- **Java keywords**

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Special values

- In addition to the keywords, Java reserves the following:
  - **true, false, null.**
- These are **values** defined by Java.
- We should not use these words for the names of variables, classes, and so on.

REFERENCE

- Herbert Schildt, Java: The Complete Reference, 8/e, Tata McGraw Hill, 2011.