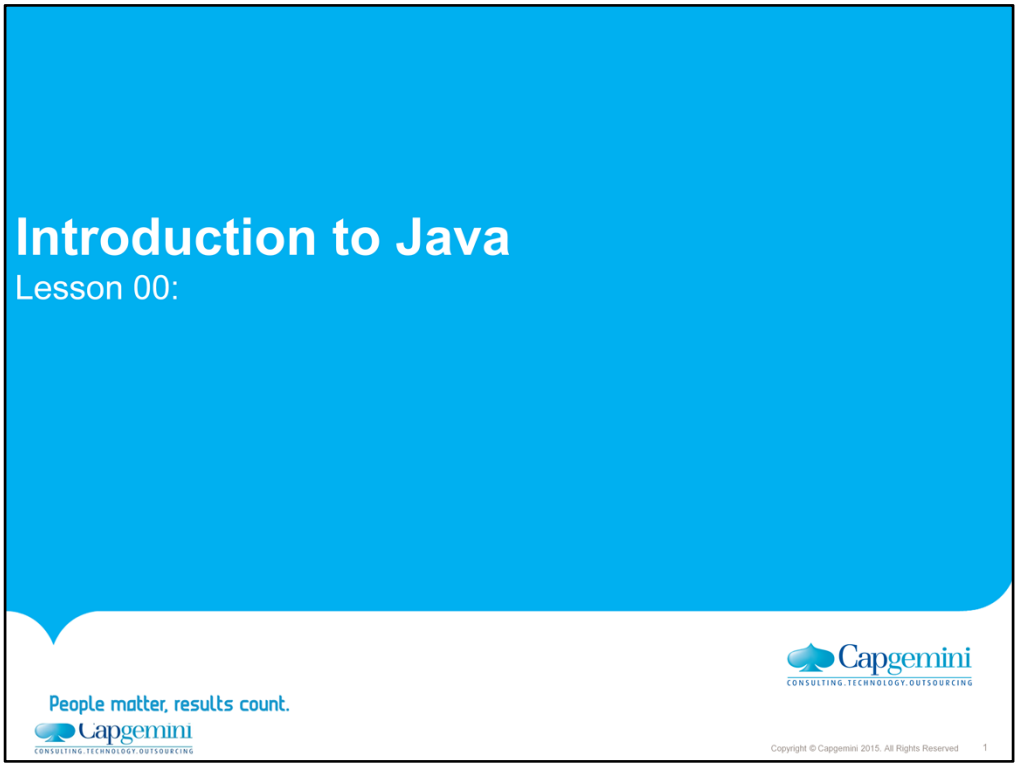


Document History

Date	Course Version No.	Software Version No.	Developer / SME	Change Record Remarks
12-Oct–2009	2.0	1.5	Anitha, Habib & Mahima	Revamped from J2SE 1.4 to J2SE 1.5
27-Oct-2009	3.0	1.5	CLS Team	Review
4- Jul-2011	4.0	1.5	Shrilata T	Changes in material made based on integration process

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 2

Course Goals and Non Goals

- Course Goals
  - Implementing OOPs features in Java
  - Developing Java Desktop Applications and Java Applets
  - Core JDK 1.5 API including JDBC
  - Processing XML document using JAXP
- Course Non Goals
  - Developing Enterprise or Web based applications

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 3

Pre-requisites

- OOPs
- DBMS/SQL
- HTML
- XML

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

Intended Audience

- Developers/Programmers

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 5

Day Wise Schedule

- Day 1
  - Lesson 1: Introduction to Java
  - Lesson 2: Eclipse 3.3 as an IDE
  - Lesson 3: Language Fundamentals
- Day 2
  - Lesson 3 (Cont.): Language Fundamentals

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 6

Table of Contents

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 7

References

- Books:
  - Java, The Complete Reference; by Herbert Schildt
  - Thinking in Java; by Bruce Eckel
  - Java 2 (Jdk 5 Ed.) Programming Black Book 2006 Ed.; by Steve
- Websites:
  - Java home page: http://java.sun.com/
  - JDK 1.5 documentation: http://java.sun.com/j2se/1.5.0/docs/

Next Step Courses

- Servlets
- JSP

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

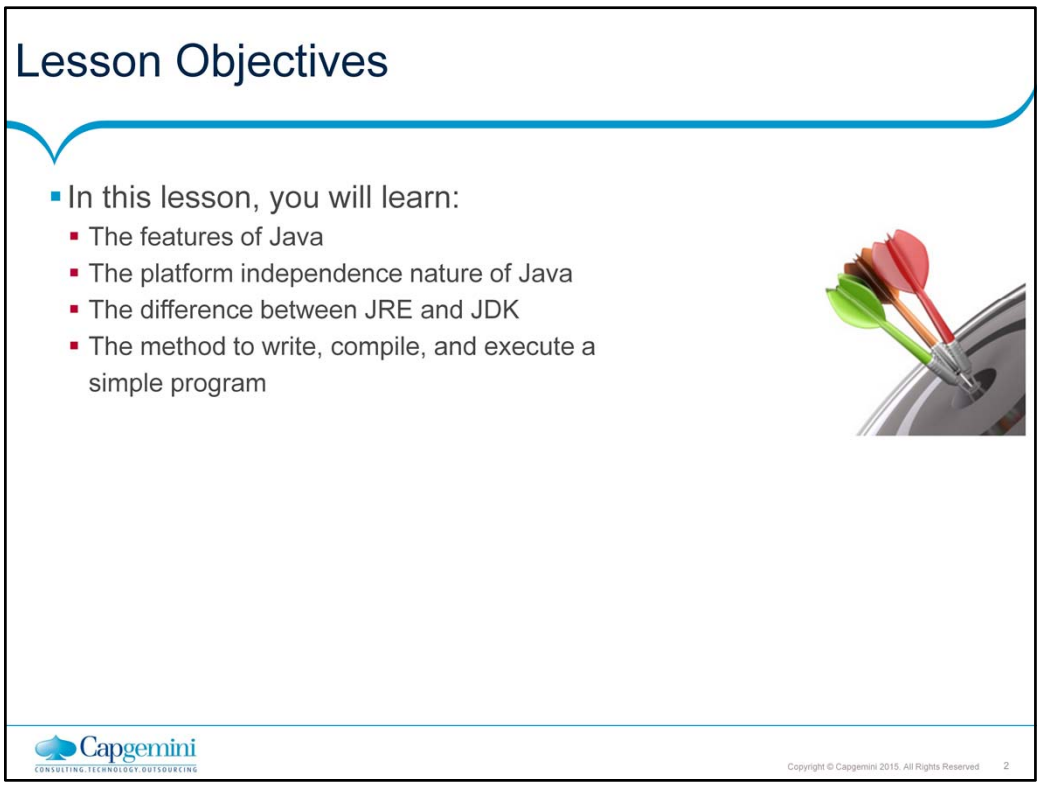
Other Parallel Technology Areas

- C ++
- .Net Framework
- C#.Net
- Visual Basic.Net

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

Introduction to Java

Lesson 1

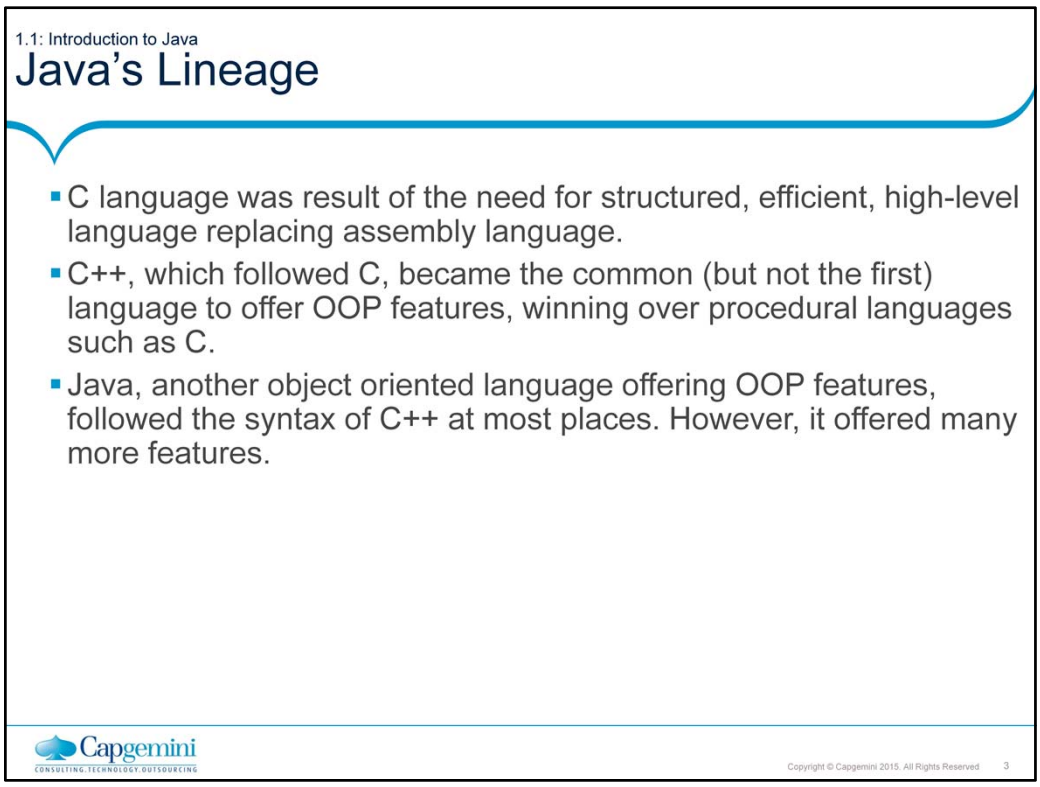


Lesson Objectives:

This lesson discusses about the features of Java, the various terminologies used in Java.

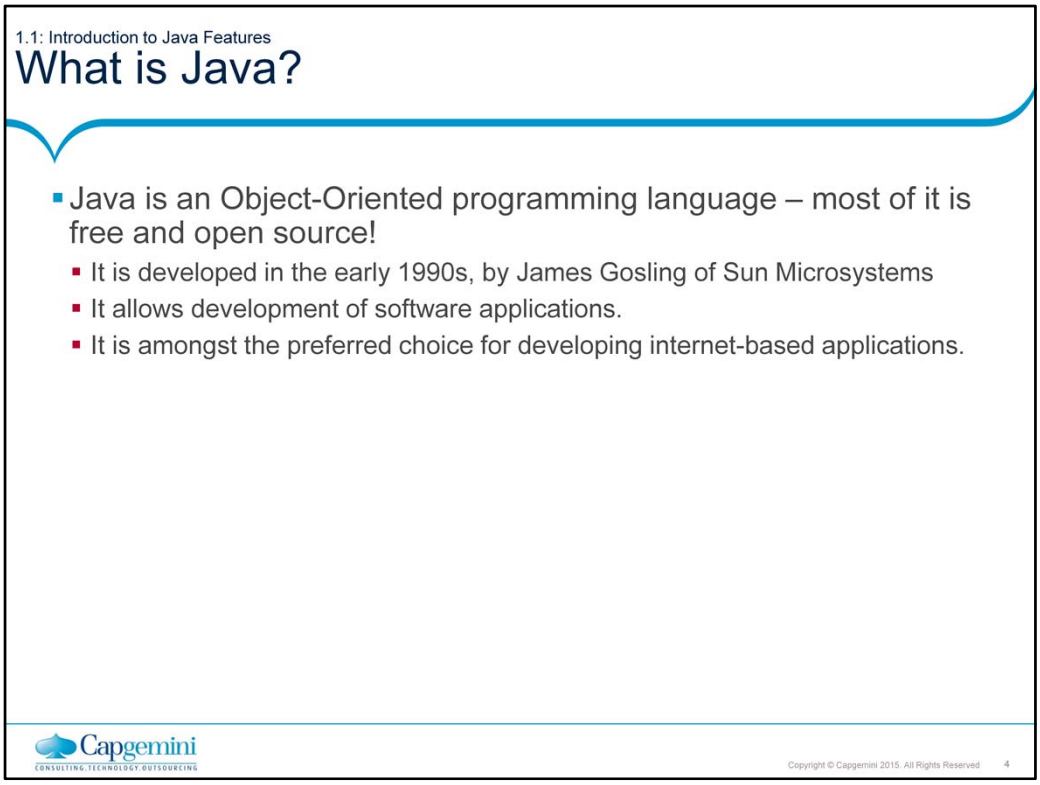
Lesson 1: Introduction to Java

- 1.1: Introduction to Java
- 1.2: Features of Java
- 1.3: Evolution in Java
- 1.4: Developing software in Java
  - 1.4.1: Platform Independence
  - 1.4.2: The JVM
  - 1.4.3: Difference between JDK and JRE



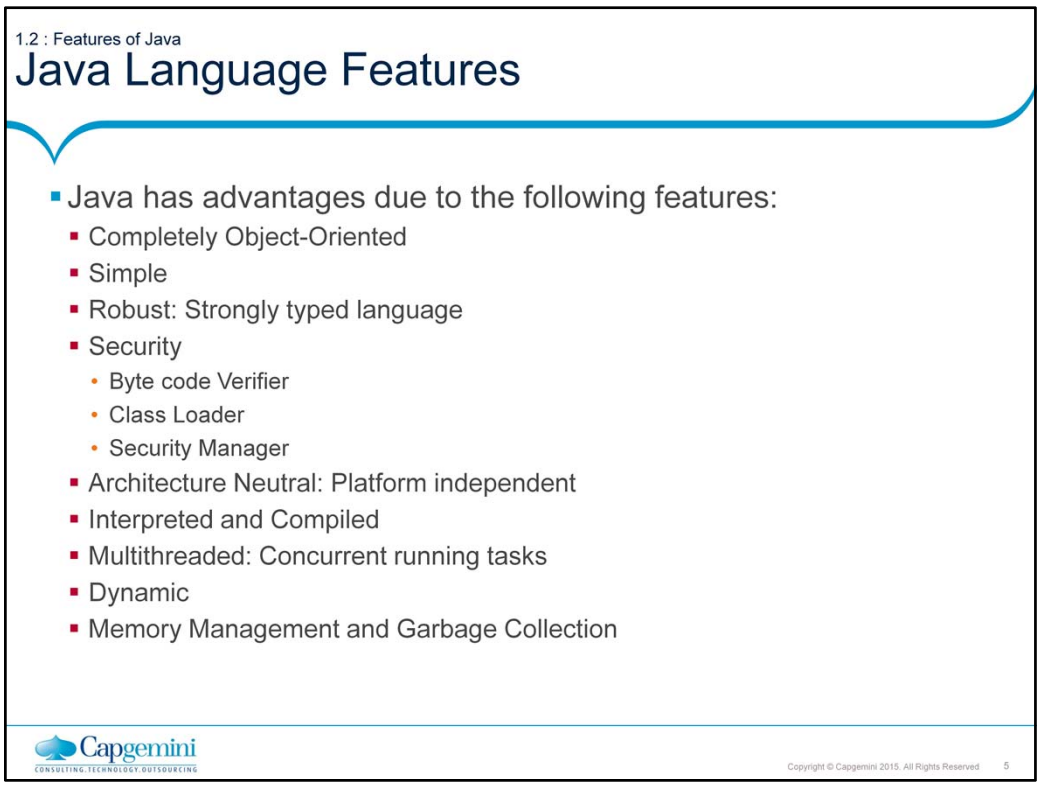
Introduction to Java:

- To understand Java, a new age Internet programming language, first it is essential to know the forces that drove to the invention of this kind of language. The history starts from a language called as **B,** which led to a famous one **C** and then to **C++**. However, the jump from C to C++ was a major development, as the whole approach of looking at an application changed from **procedural way** to **object oriented way**. The languages like, Smalltalk, were already using the Object Oriented features.
- By the time C++ got the real acknowledgement from the industry, there was a strong need for a language which creates architecture neutral, platform independent, and portable software. The reason behind this particular need was the Embedded software market, which runs on variety of consumer electronic devices. Hence a project called as "OAK" was started at Sun Microsystems.
- The second force behind this is WWW (World Wide Web). Now on WWW, most of computers over the Internet are divided into three major architectures namely Intel, Macintosh, and Unix. Thus, a program written for the Internet has to be a portable program, so that it runs on all the available platforms.
- All this led to the development of a new language in 1995, when they renamed the "OAK" language to "JAVA".



Introduction to Java Features – What is Java?

- Java programming language is a high level programming language offering Object-Oriented features. James Gosling developed it at Sun Microsystems in the early 1990s. It is on the lines of C/C++ syntax, however, it is based on an object model. Sun offers much of Java as free and open source. Today we have the Java version 6 in the market.
- The Java programming language forms a core component of Sun's Java Platform. By platform, we mean the mix of hardware and software environment in which a program executes – such as MS Windows and Linux. Sun's Java is a "software-only" platform which runs on top of other hardware platforms. We will learn more about this on subsequent slides.
- Today, Java has become an extremely popular language. The platform is amongst the preferred choices for developing internet based applications. Why is it so? Let us explore!



Features of Java:

Java is completely object oriented. C++, being more compatible to C, allows code to exist outside classes too. However in Java, every line of code has to belong to some or other class. Thus it is closer to true object oriented language.

Java is simpler than C++, since concepts of pointers or multiple inheritance do not exist.

Let us discuss these features in detail.

Object-Oriented:

To stay abreast of modern software development practices, Java is Object-Oriented from the ground up. Many of Java's Object-Oriented concepts are inherited from C++, the language on which it is based, plus concepts from other Object-Oriented languages as well.

Simple:

Java omits many confusing, rarely used features of C++. There is no pointer level programming or pointer arithmetic. Memory management is automatic. There are no header files, structures, unions, operator overloading, virtual base classes, and multiple inheritance.

Robust:

Java programs are reliable. Java puts a lot of emphasis on early checking for potential problems, dynamic checking and eliminating situations that are error-prone. Java has a pointer model that eliminates possibility of overwriting memory and corrupting data.

Features of Java (contd.):

Security:

Java is intended to be used in networked / distributed environments. Thus a lot of emphasis is placed on security. Normally two things affect security:

- confidential information may be compromised, and
- computer systems are vulnerable to corruption or destruction by hackers

Java's security model has three primary components:

- **Byte code Verifier:** The byte code verifier ensures the following:
  - the Java programs have been compiled correctly,
  - they will obey the virtual machine's access restrictions, and
  - the byte codes will not access private data when they should not
- **Class Loader:** When the loader retrieves classes from the network, it keeps classes from different servers separate from each other and from local classes. Through this separation, the class loader prevents a class that is loaded off the network from pretending to be one of the standard built-in classes, or from interfering with the operation of classes loaded from other servers.
- **Security Manager:** It implements a security policy for the VM. The security policy determines which activities of the VM is allowed to perform and under what circumstances, operation should pass.

Architecture-Neutral:

A central issue for the Java designers was of code longevity and portability. One of the main problems facing programmers is that there is no guarantee that when you write a program today, it will run tomorrow — even on the same machine. Operating system upgrades, processor upgrades, and changes in core system resources can all combine to make a program malfunction. The Java designers made several hard decisions in the **Java language** and the **Java Virtual Machine** (JVM) in an attempt to alter this situation. Their goal was "write once; run anywhere, any time, forever". To a great extent, this goal was accomplished.

Platform-Independent:

This refers to a program's capability of moving easily from one computer system to another. Java is platform-independent at both the source and the binary level.

- At the source level, Java's primitive data types have consistent sizes across all development platforms.
- Java binary files are also platform-independent and can run on multiple platforms without the need to recompile the source because they are in the form of byte codes.

Features of Java (contd.):

Interpreted and Compiled:

Java programs are compiled into an intermediate byte code format, which in turn will be interpreted by the VM at run time. Hence, any Java program is checked twice before it actually runs.

Multithreaded:

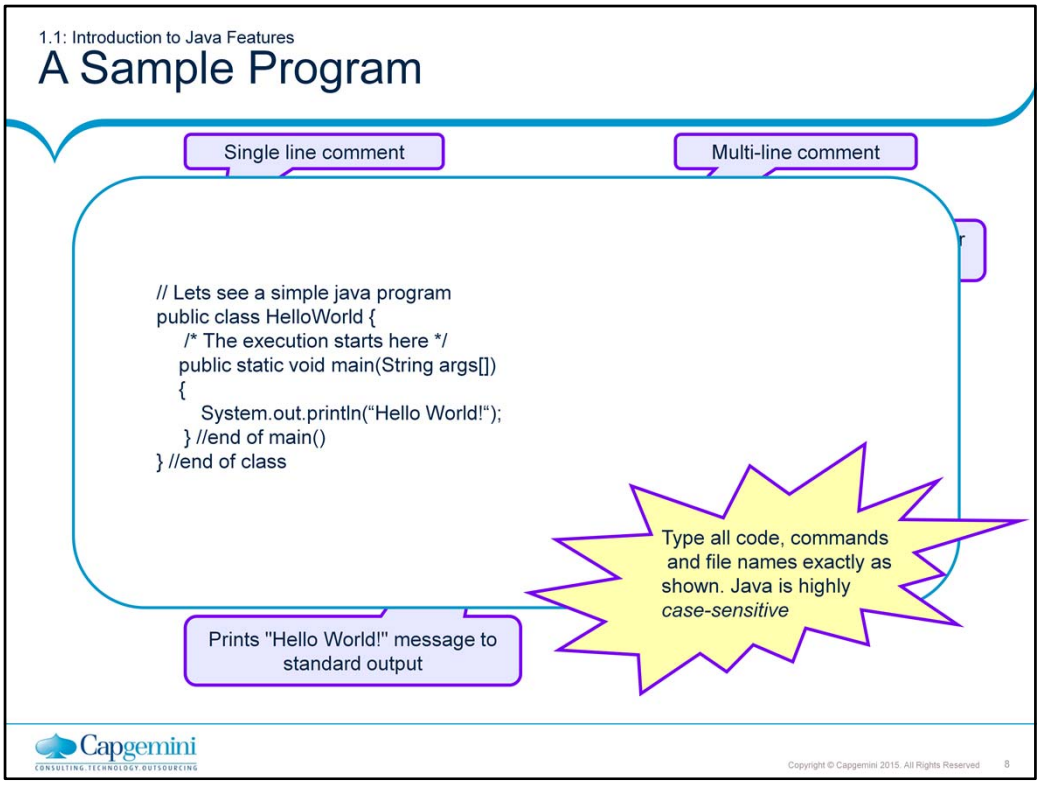
A multi-threaded application can have several threads of execution running independently and simultaneously. These threads may communicate and co-operate. To the user it will appear to be a single program. Java implements multithreading through a part of its class library. However, Java also has language constructs to make programs thread-safe.

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. This is crucial to the robustness of the applet environment, in which small fragments of bytecode may be dynamically updated on a running system.

Memory Management and Garbage Collection:

Java manages memory de-allocation by using garbage collection. Temporary memory is automatically reclaimed after it is no longer referenced by any active part of the program. To improve performance, Java's garbage collector runs in its own low-priority thread, providing a good balance of efficiency and real-time responsiveness.

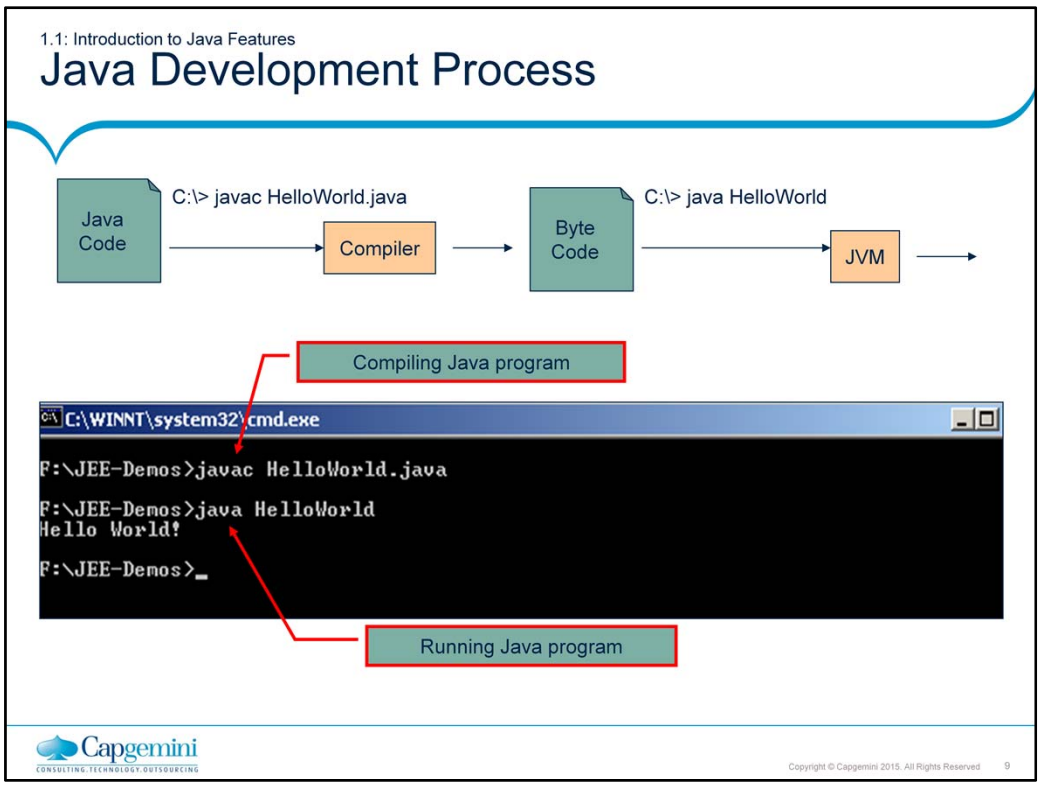


<u>Introduction to Java Features – A Sample Program</u>:

- Here is our first Java program..

- Let us have a closer Look at the "Hello World!" program:
  - **class HelloWorld** begins the class definition block for the "Hello World!" program. Every Java program must be contained within a "class" block.
  - **public static void main(String[] args) { …}** is the entry point for your application. Subsequently, it invokes all the other methods required by the program.

- This program when executed will display "Hello World" on the screen. We shall see this in the next slide.



<u>Introduction to Java Features – Java Development Process</u>:

The Java Development Process involves the following steps:

1. Write the Java code in a text file with a .java extension, namely "HelloWorld.java". By convention, source file names are named by the public class name they contain.
2. At the command line, compile the code using the Java compiler (javac). The javac compiler converts the source into Byte Codes and stores it in a file having .class extension. What is special about byte codes? Unlike traditional compilers, javac does not produce processor specific native code. Instead it generates code which is in the language of JVM (Java Virtual Machine).
   Note: To have access to the **javac** and the **java** commands, you must set your path first. To do so, you may type the following at the command prompt:
   Set path=<your java-home directory>\bin
   For example: *Set path=C:\j2sdk1.4.1_07\bin*
   We can also set the environment variables (path and classpath) through the Control Panel.
3. To run this program, use the Java command with class file name as the command parameter as shown on the above slide. The output of the program would, of course, be "Hello World!"

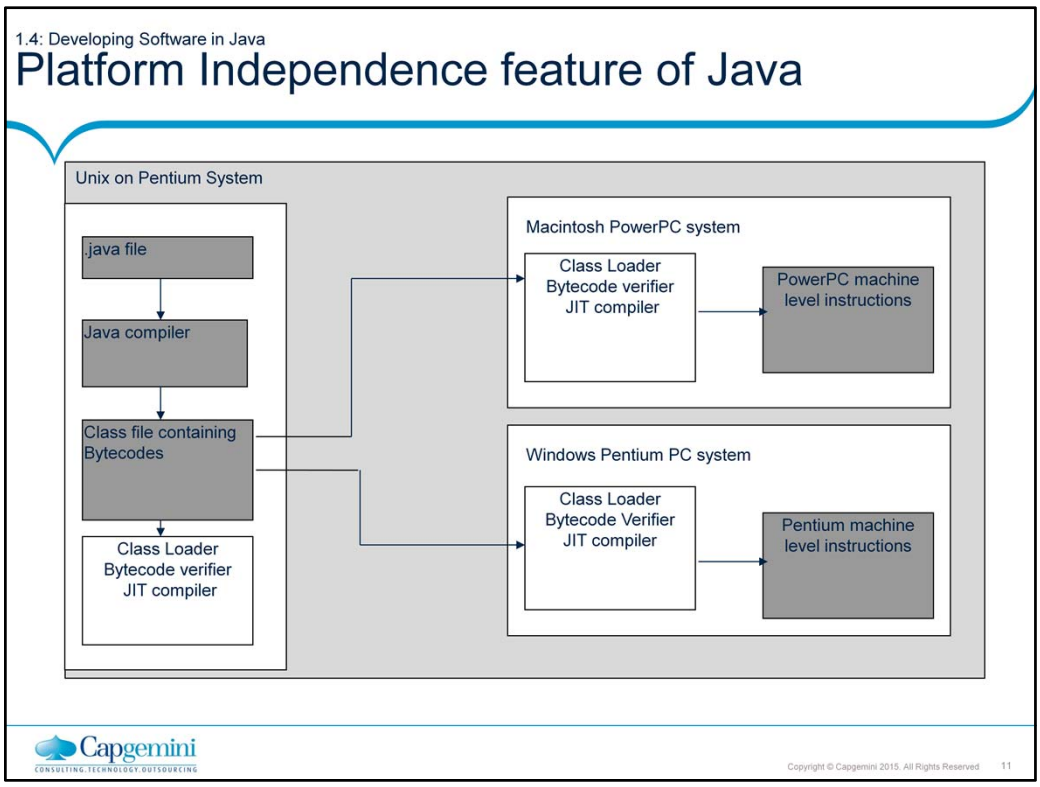
1.1: Introduction to Java Features

Demo

- Creating and executing the “Hello World!” program

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 10



Platform Independence:

•The figure illustrates how platform independence is achieved using Java. Once you write Java code on a platform and run it through Java Compiler, the class file containing byte codes is obtained.

•Different JVMs are available for different platforms. So the JVM for Unix on Pentium will be different from the JVM for Mac or for Windows. Each of these JVMs take the same input, namely the Class File, and produce the machine level instructions for the respective platforms.

•One common grouse among developers is that Java programs take longer to execute because the compiled bytecodes are *interpreted* by the JVM. The Java just-in-time (JIT) compiler, compiles the bytecode into platform-specific executable code (**native code**) that is immediately executed, thus speeding up execution! Traditional native code compilers run on the developer's machine and are used by programmers, and produce non-portable executables. JIT compilers run on the user's machine and are transparent to the user. The resulting native code instructions do not need to be ported because they are already at their destination.

Platform Independence (contd.):

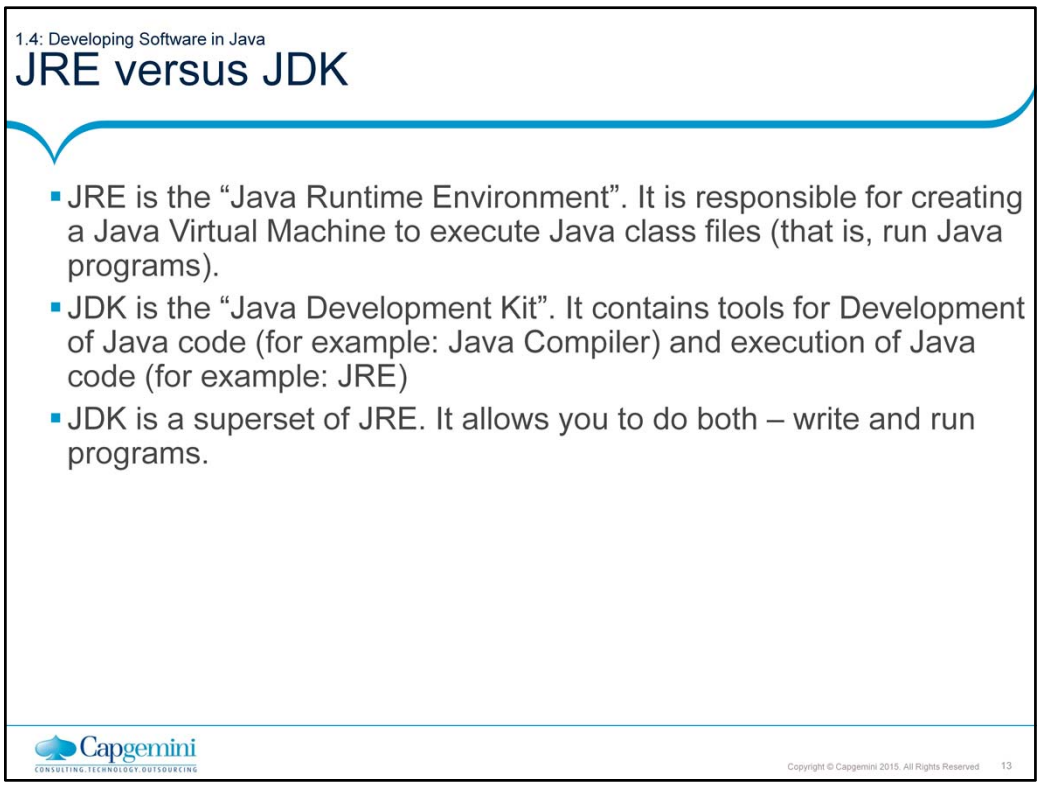
JVM:

When you compile a Java program (which usually is a simple text file with .java extension), it is compiled to be executed under VM. This is in contrast to C/C++ programs, which are compiled to be run on a real hardware platform, such as a Pentium processor running on, say Win 95. The VM itself has characteristics very much like a physical microprocessor. However, it is entirely a software construct. You can think of the VM as an intermediary between **Java programs** and the underlying **hardware platform** on which all programs must eventually execute.

- Even with the VM, at some point, all Java programs must be resolved to a particular underlying hardware platform. In Java, this resolution occurs within each particular VM implementation. The way this works is that Java programs make calls to the VM, which in turn routes them to appropriate native calls on the underlying platform. It is obvious that the **VM itself** is very much **platform dependent**.

How does the JIT compiler work?

- The VM instead of calling the underlying native operating system, it calls the JIT compiler. The JIT compiler in turn generates native code that can be passed on to the native operating system for execution. The primary benefit of this arrangement is that the JIT compiler is completely transparent to everything except VM. The neat thing is that a JIT compiler can be integrated into a system without any other part of the Java runtime system being affected.
- The integration of JIT compilers at the VM level makes JIT compilers a legitimate example of component software. You can simply plug in a JIT compiler and reap the benefits with no other work or side effects.
- A Java enabled browser contains its own VM. Web documents that have embedded Java applets must specify the location of the main applet class file. The Web browser then starts up the VM and passes the location of the applet class file to the class loader. Each class file knows the names of any additional class files that it requires. These additional class files may come from the network or from client machine. Supplement classes are fetched only if they are actually going to be used or if they are necessary for the verification process of the applets.



Difference between JRE and JDK:

- The **Java Development Kit (JDK)** is a superset which includes Java Compilers, Java Runtime Environments (JRE), Development Libraries, Debuggers, Deployment tools, and so on. One needs JDK to develop Java applications. We have different versions that include JDK 1.2, JDK 1.4, and so on.
- The **Java Runtime Environment (JRE)** is an implementation of JVM that actually executes the Java program. It is a subset of JDK. One needs JRE to execute Java applications.

Summary

- In this lesson, you have learnt:
  - Features of Java and its different versions
  - How Java is platform Independent
  - Difference between JRE and JDK
  - Writing, Compiling, and Executing a simple program

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 14

Review Questions

- Question 1: A program written in the Java programming language can run on any platform because...
  - **Option 1:** The JIT Compiler converts the Java program into machine equivalent
  - **Option 2:** The Java Virtual Machine1(JVM) interprets the program for the native operating system
  - **Option 3:** The compiler is identical to a C++ compiler
  - **Option 4:** The APIs do all the work

- Question 2: Java Compiler compiles the source code into ___ code, which is interpreted by ___ to produce Native Executable code.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 15

Review Questions

- Question 3: Which of the following are true about JVM?
  - **Option 1:** JVM is an interpreter for byte code
  - **Option 2:** JVM is platform dependent
  - **Option 3:** Java programs are executed by the JVM
  - **Option 4:** All the above is true

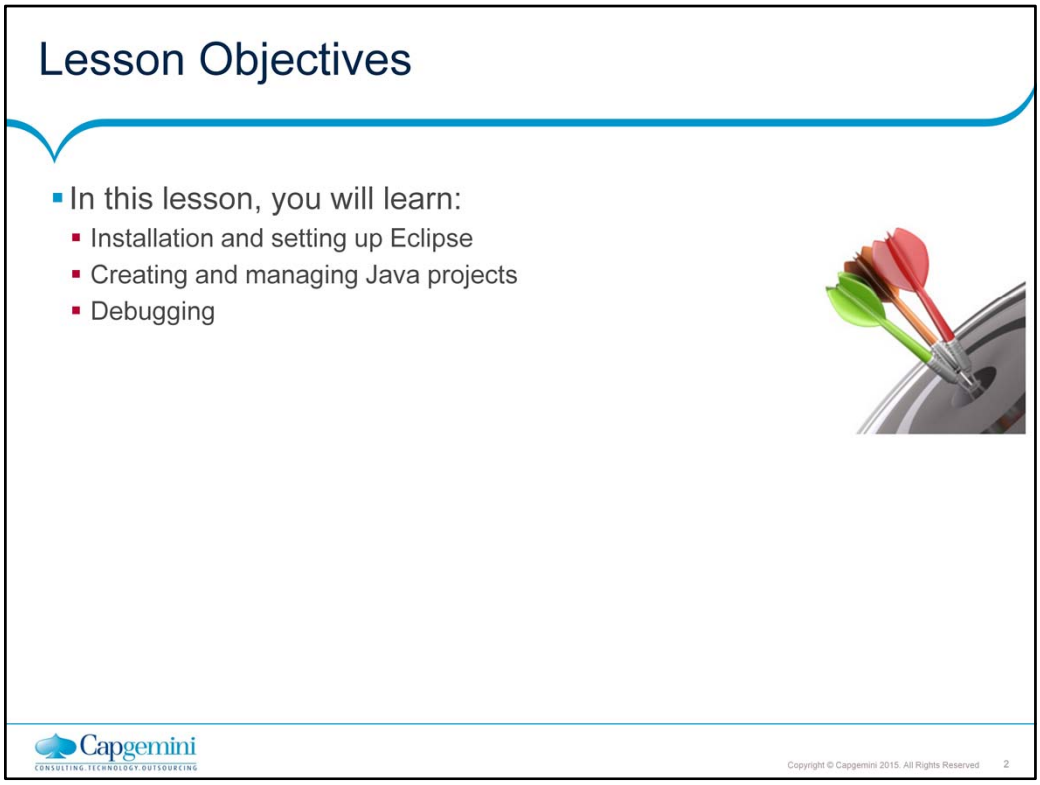
- Question 4 : ____ allows a Java program to perform multiple activities in parallel.
  - **Option 1:** Java Beans
  - **Option 2:** Swing
  - **Option 3:** Multithreading
  - **Option 4:** None of the above

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

 16

Introduction to Java

Lesson 2: Eclipse 3.5 as an IDE



Lesson Objectives:

This lesson introduces you to Eclipse an IDE

Lesson 2: Eclipse3.5 as an IDE

- 2.1: Installation and setting up Eclipse
- 2.2: Introduction to Eclipse IDE
- 2.3: To create and manage Java projects
  - 2.3.1: Debugging your Java Program
- 2.4: Miscellaneous options
  - 2.4.1: Creating Jar files
  - 2.4.2: Verifying JRE installation
  - 2.4.3: Creating a Jar file
  - 2.4.4: Setting Classpath
  - 2.4.5: Passing Command line arguments
  - 2.4.6: Import and Export Options
  - 2.4.7: Automatic Build/Manual Build options
  - 2.4.6: Tips and Tricks

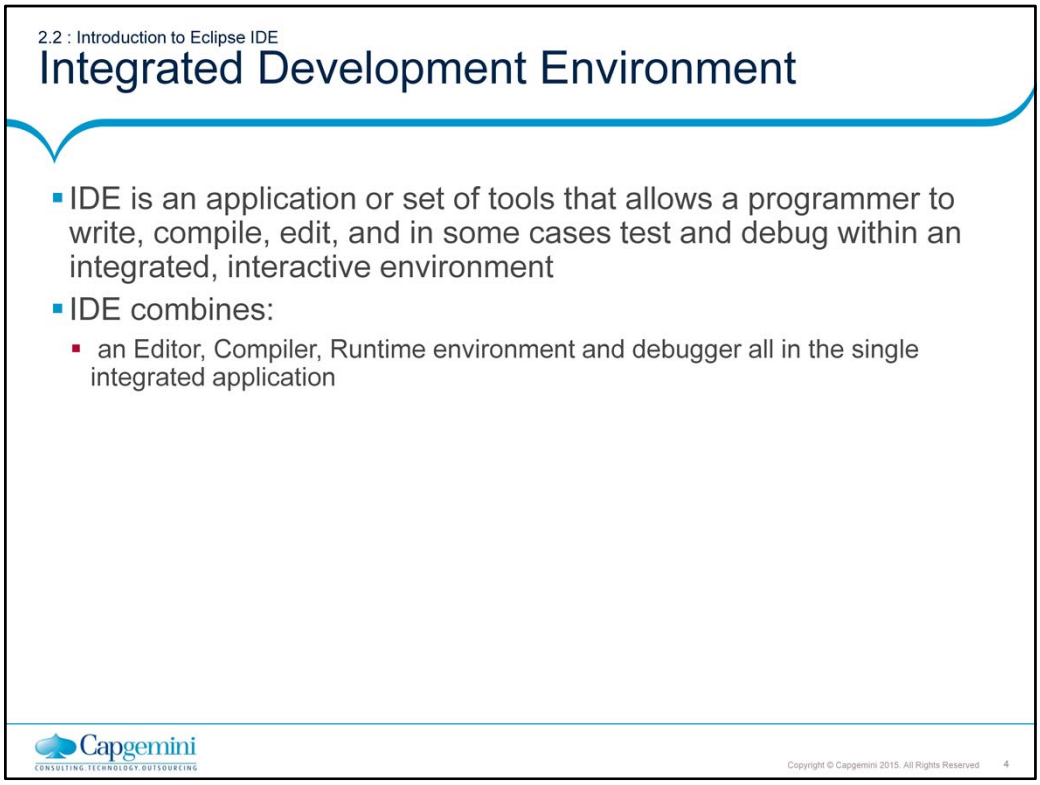
2.1: Installation and Setting up Eclipse

Installing Eclipse 3.3

- You need to follow the given steps to install Eclipse 3.5:
  - Download Eclipse-SDK zip file
  - Unpack the Eclipse SDK into the target directory
    - For example: c:\eclipse3.5
  - To start Eclipse, go to the eclipse subdirectory of the folder in which you extracted the zip file
    (for example: c:\eclipse3.5\eclipse) and run eclipse.exe

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

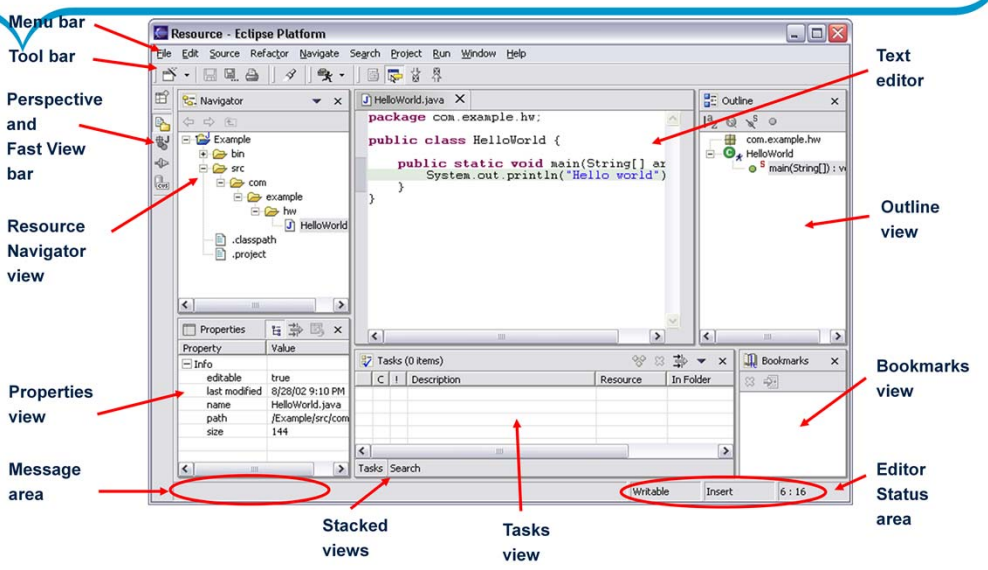
 3



What is an IDE?

- The **Eclipse Project** is an open source software development project dedicated to providing a robust, full-featured, commercial-quality, industry platform for the development of highly integrated tools and rich client applications. Eclipse runs on Windows, Linux, Mac OSX, Solaris, AIX and HP-UX. Eclipse is actually a generic application platform with a sophisticated plug in architecture - the Java IDE is just one set of plugins. There is an active community of third party Eclipse plugin developers, both open source and commercial. Our objective is to code Java programs faster with Eclipse 3.5 as an IDE.
- Eclipse3.3 features include the following:
  - Creation and maintenance of the Java project
  - Developing Packages
  - Debugging a java program with variety of tools available
  - Running a Java program
- Developing the Java program will be easier as Eclipse editor provides the following:
  - Syntax highlighting
  - Content/code assist
  - Code formatting
  - Import assistance
  - Quick fix

Workbench Terminology

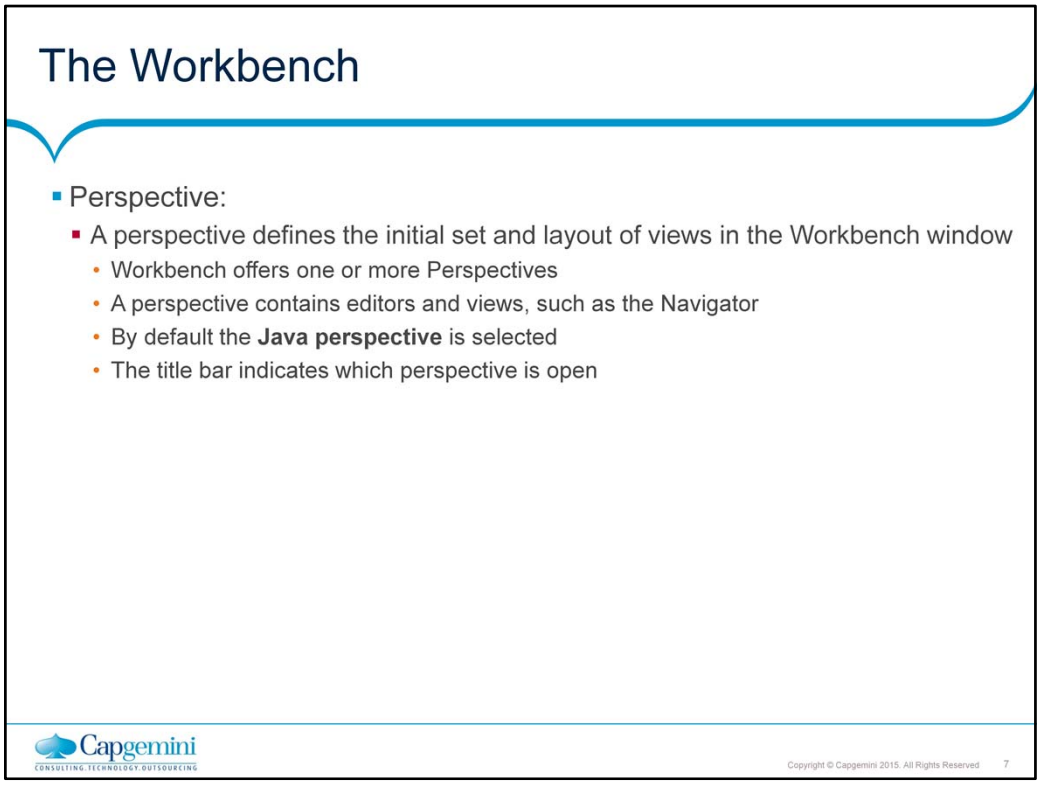


The Workbench

- The term “Workbench” refers to the desktop development environment
- It allows you to select the Workspace
- A Workbench consists of the following:
  - perspectives
  - views
  - editors

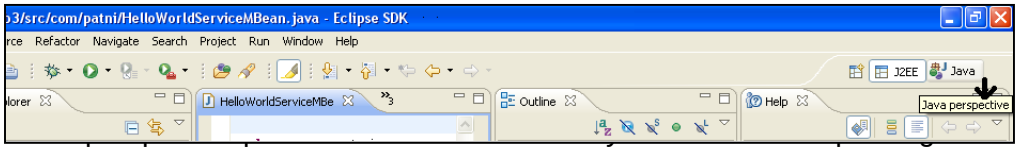
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 6



The Workbench:
The term **Workbench** refers to the desktop development environment.

Perspectives:
Each Workbench window contains one or more perspectives. Perspectives contain **views** and **editors** and control what appears in certain **menus** and **tool bars**. They define visible **action sets**, which you can change to customize a perspective. You can save a perspective that you build in this manner, making your own custom perspective that you can open again later. By default the Java perspective is selected.



specific type of task or works with specific types of resources.
For example: The **Java perspective** combines views that you would commonly use while editing Java source files, while the **Debug perspective** contains the views that you would use while debugging Java programs. As you work in the Workbench, you will probably switch perspectives frequently.

The Workbench

- View:
  - It is the visual component within the Workbench
  - It is used to navigate a hierarchy of information or display properties for the active editor

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

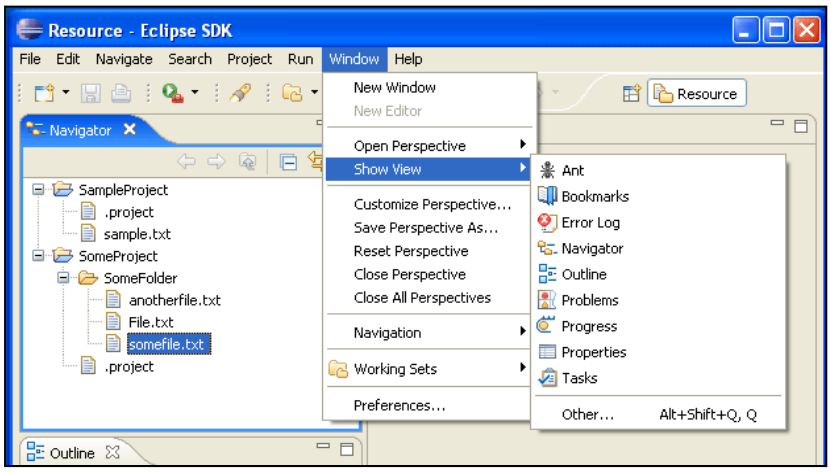
The Workbench:

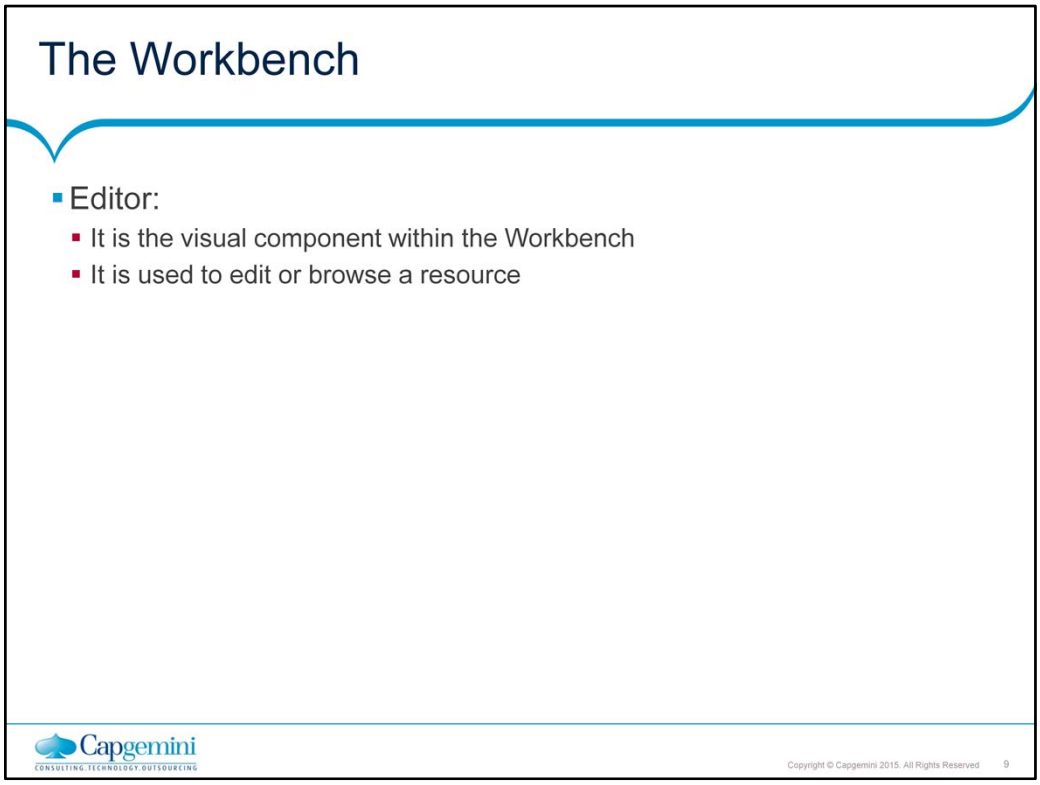
View:

Views support editors and provide alternative presentations as well as ways to navigate the information in your Workbench.

For example: The Project Explorer and other navigation views display projects and other resources that you are working with.

Perspectives offer pre-defined combinations of views and editors. To open a view that is not included in the current perspective, select **Window → Show View** from the main menu bar.



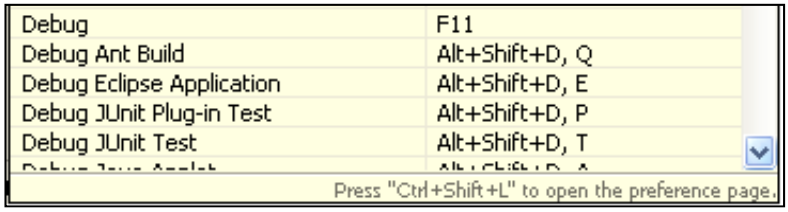


The Workbench:
Editor: Most perspectives in the Workbench comprise an **editor area** and one or more **views**. You can associate different editors with different types of files.
For example: When you open a file for editing by double-clicking it in one of the navigation views, the associated editor opens in the Workbench.

If there is no associated editor for a resource, then the Workbench attempts to launch an external editor outside the Workbench.
For example: Suppose you have a .doc file in the Workbench, and Microsoft Word is registered as the editor for .doc files in your operating system. Then opening the file will launch Word as an OLE document within the Workbench editor area.

The Workbench:

- The Workbench menu bar and toolbar will be updated with options for Microsoft Word).
- Any number of editors can be open at once. However, only one can be active at a time. The main menu bar and toolbar for the Workbench window contain operations that are applicable to the active editor.
- Useful Tips and Tricks related with Work Bench:
  - **View all Keyboard shortcuts:** While working with your favorite editors and views in Eclipse, just press **Ctrl+Shift+L** to see a full list of the currently available key bindings.

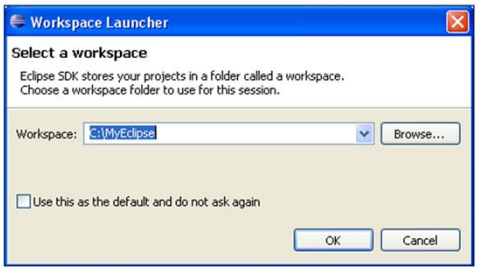


  - **Importing Files:** You can quickly import files and folders into your workspace by dragging them from the file system (for example: from a Windows Explorer window) and dropping them into the Project Explorer view. The files and folder are always copied into the project. The originals are not affected. Copy and paste also work.
  - **Exporting Files:** You can drag files and folder from the Project Explorer view to the file system (e.g., to a Windows Explorer window) to export the files and folders. The files and folder are always copied. However, the workspace resources are not affected. Copy and paste also work.
  - **Switch Workspace:** Instead of shutting down eclipse and restarting with a different workspace, you can instead use **File → Switch Workspace**. From here you can either open previous workspaces directly from the menu or you can open the **workspace chooser** dialog to choose a new one.

2.3: Creating and Managing Java Projects

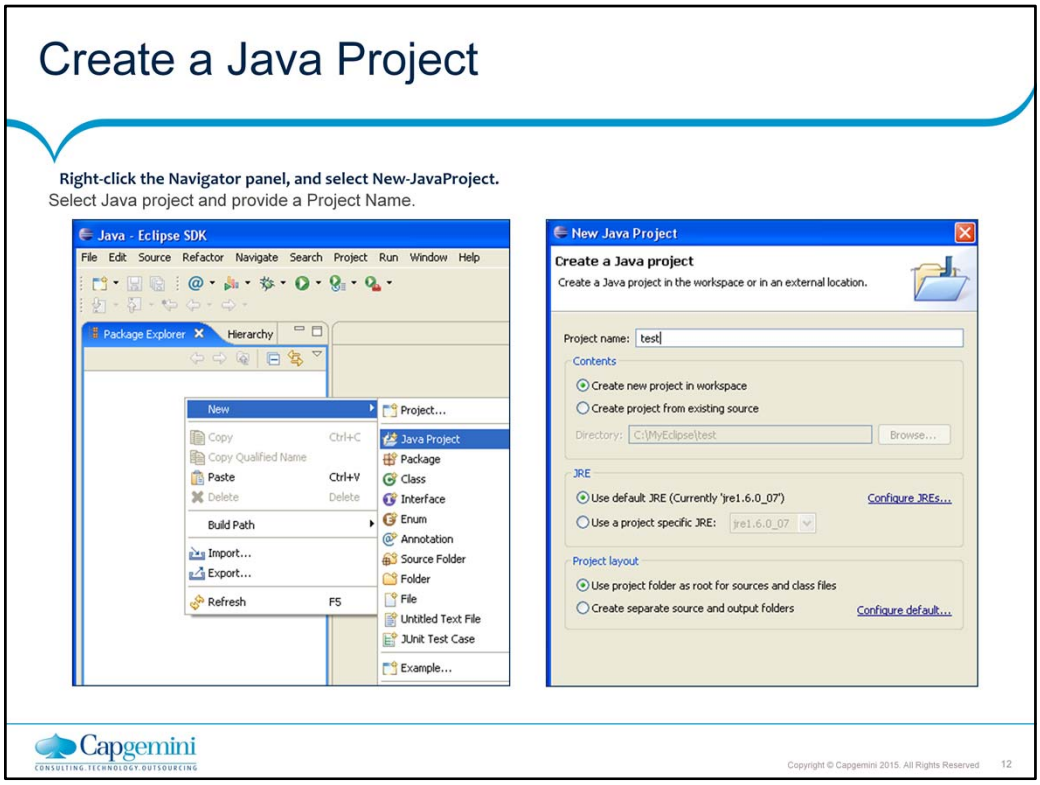
Create Workspace

- You need to follow the given steps to create a workspace:
  - Start up Eclips
  - Supply a path to a new folder which will serve as your workspace
  - The workspace is a folder which Eclipse uses to store your source code



Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 11



<u>Creating and Managing Java Projects</u>:

Create a Java Project:

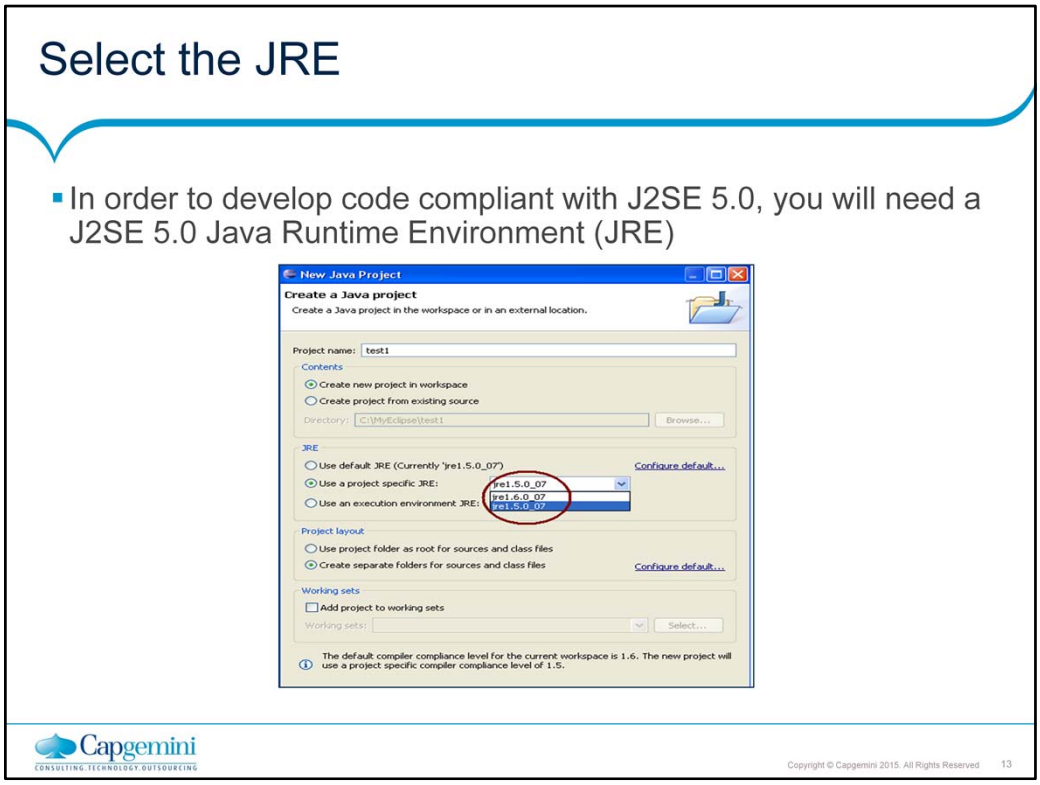
- When Eclipse starts, you will see the Welcome page. Close the Welcome page.
- Right-click in the Navigator panel, and select **New → JavaProject** and provide a Project name.

Note:

- The name of the project is test. It is created in the Workspace C:\MyEclipse
- The Project uses jre1.6.0_07. The same folder will be used for storing both the source files and the class files.

The Java project can now include the following:

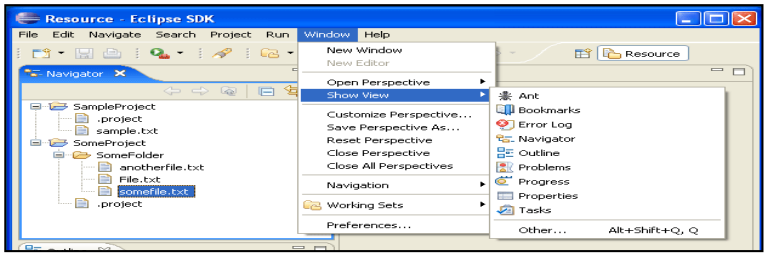
- Class
- Package
- Interface
- Source folder
- Folder
- File
- Junit Test Case
- Other - which may include other resources as text files



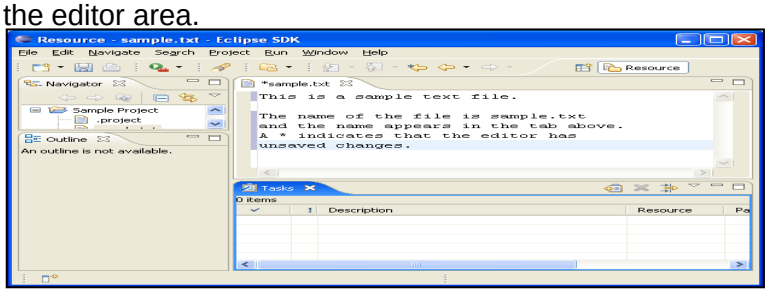
<u>Creating and Managing Java Projects</u>:

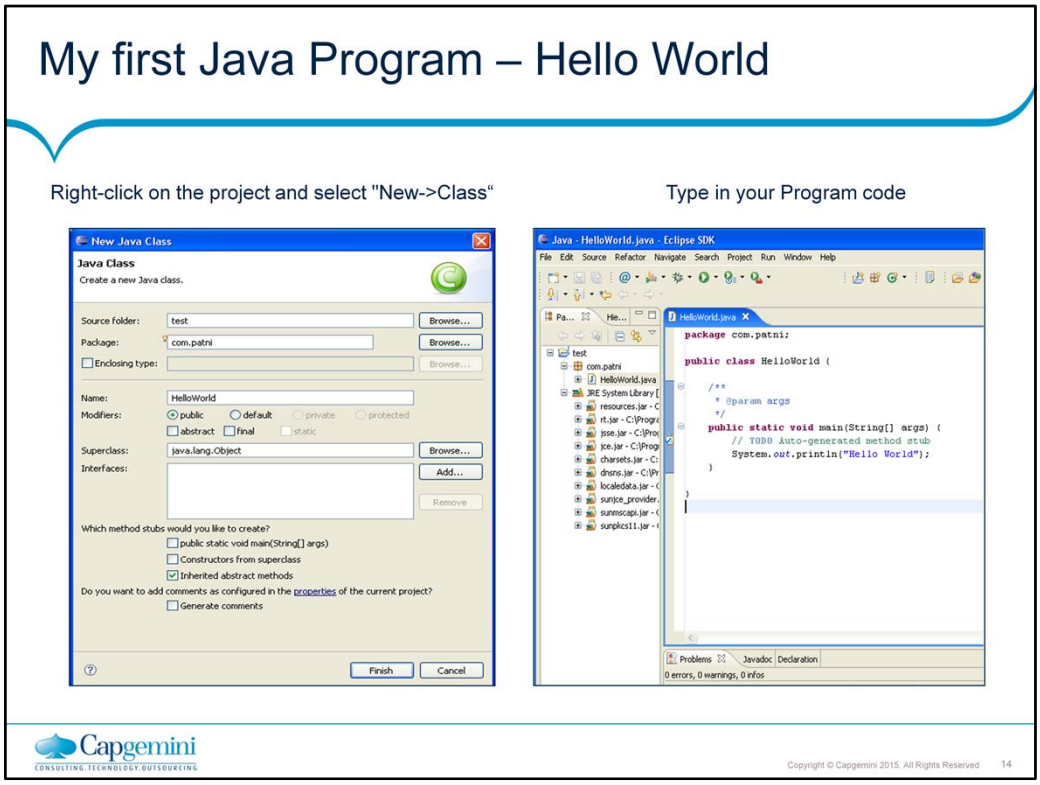
Select the JRE:

- To use the new **J2SE 5.0** features, you must be working on a project that has a **5.0 compliance level** enabled and has a **5.0 JRE**. New projects will automatically get 5.0-compliance while choosing a 5.0 JRE on the first page of the New Java Project wizard.



- Depending on the type of file that is being edited, the appropriate editor is displayed in the editor area.
  For example: If a .TXT file is being edited, a text editor is displayed in the editor area.





<u>Creating and Managing Java Projects</u>:

My first Java Program – Hello World:

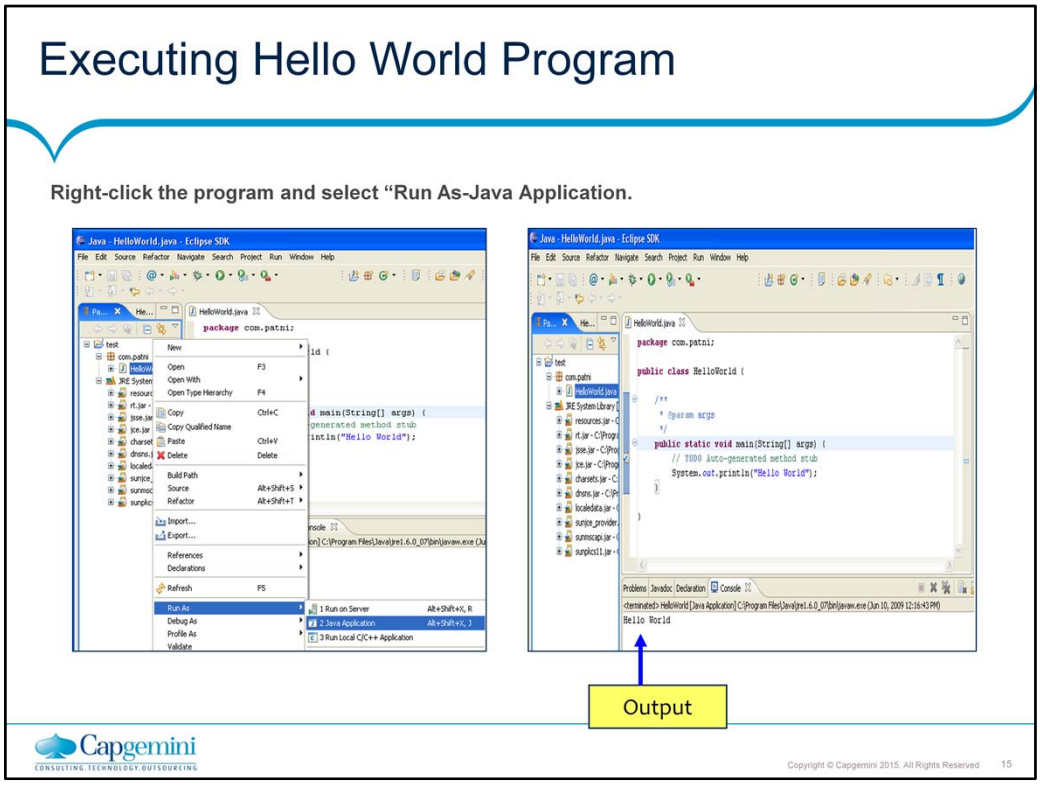
- If you want to create some new Java code, right-click the project and select "**New → Class**".
- In the dialog box created, give a name for the Java program in the **Name** textbox.
- Also notice that a package name is also given, as the usage of default package is discouraged. A package in Java is a group of classes which are often closely or logically related in some way. (The **Package chapter** is discussed later in the course).
- Eclipse will generate skeleton of the class including **default** constructor and **main ()** method.

Note:

- The package name is com.patni.
- The Java program's name is HelloWorld.
- The HelloWorld class will have the public access modifier.
- The HelloWorld class inherits from java.lang.Object.

Developing the Java program will be easier as Eclipse editor will provide:

- Syntax highlighting
- Content/code assist
- Code formatting
- Import assistance
- Quick fix



<u>Creating and Managing Java Projects</u>:

Executing Hello World Program:

- Right-click the **HelloWorld.java** in the Package Explorer, and select **Run As → Java Application**.
- The program after execution produces the output in the **Console view**.
- Running class from the Package Explorer as a Java Application uses the default settings for launching the selected class, and does not allow you to specify any arguments.

2.3: Creating and Managing Java Projects using Eclipse

Lab: Executing HelloWorld Program

- Lab 2.1: HelloWorld Program

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 16

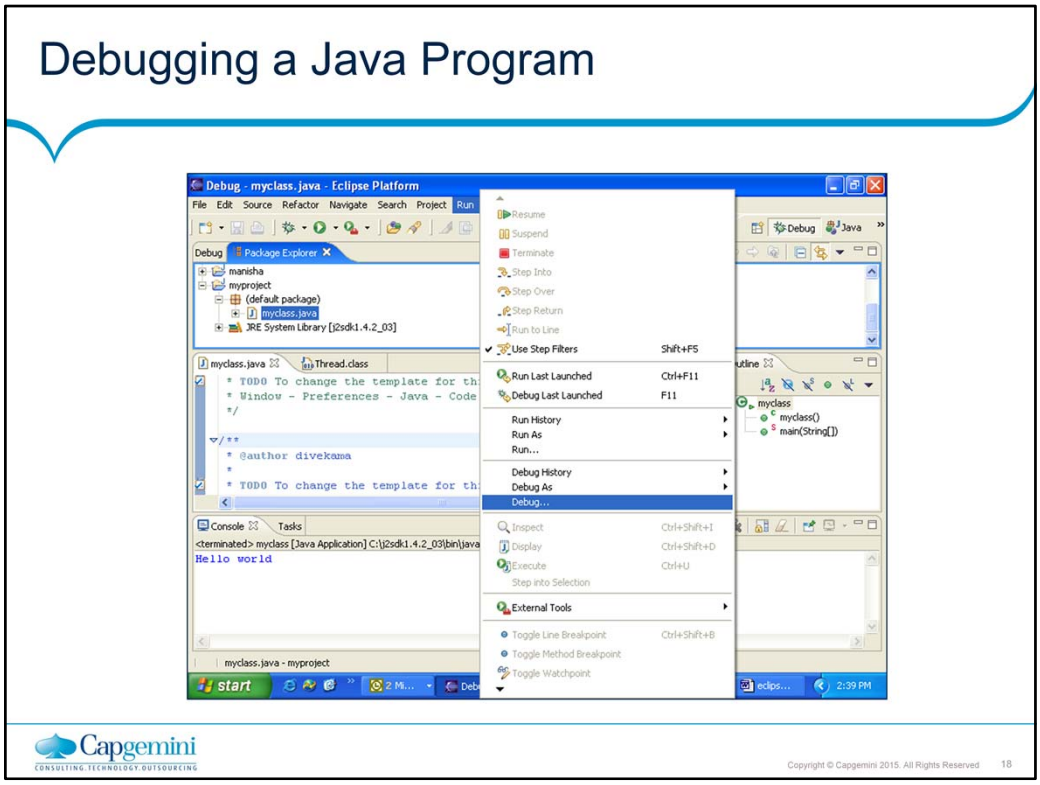
2.3: Creating and Managing Java Projects

Debugging your Java Program using Eclipse

- The Java Development Toolkit (JDT) includes a debugger that enables you to detect and diagnose errors in your programs running either locally or remotely
- The debugger allows you to control the execution of your program by employing the following:
  - setting breakpoints, suspending launched programs, stepping through your code, and examining the contents of variables

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 17

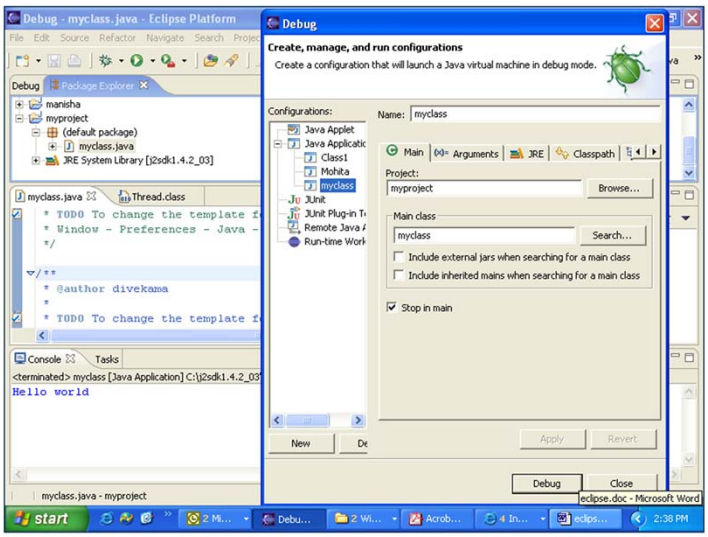


<u>Creating and Managing Java Projects</u>:

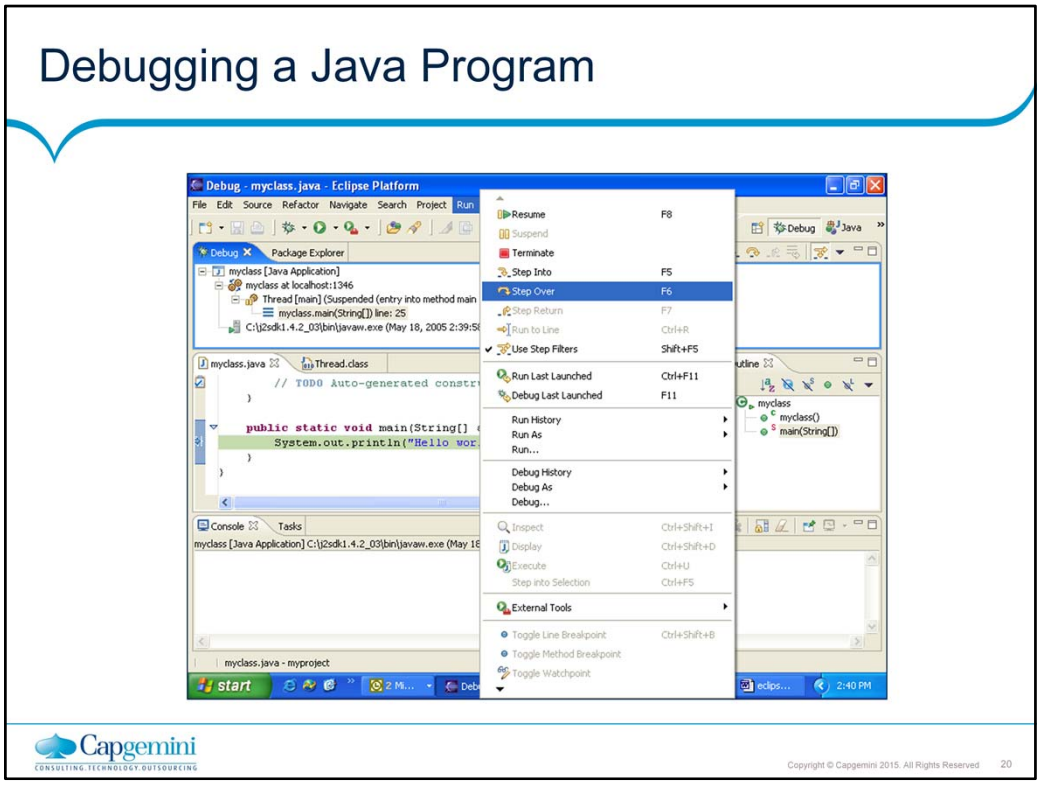
Debugging a Java Program:

- Eclipse gives you **auto-build** facility, where recompilation of the necessary Java classes is done automatically.
- To debug your Java program, select the Java source file which needs to be debugged, select **Run → Debug**. This will ask you to select an option to halt in public static void main() method, from where you may select to step into each and every function you come across or step over every function and only capture output of each function.

Specifying Debugging options



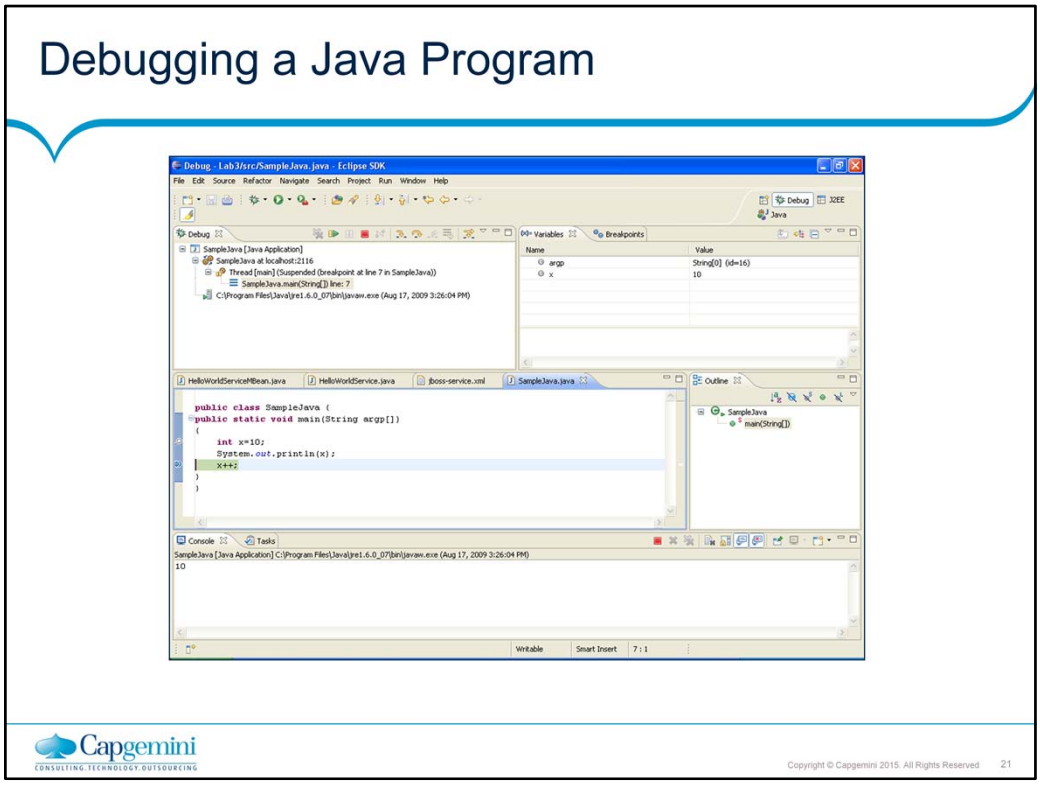
Note: Click **Debug** to start debugging process.



<u>Creating and Managing Java Projects</u>:

Debugging a Java Program:

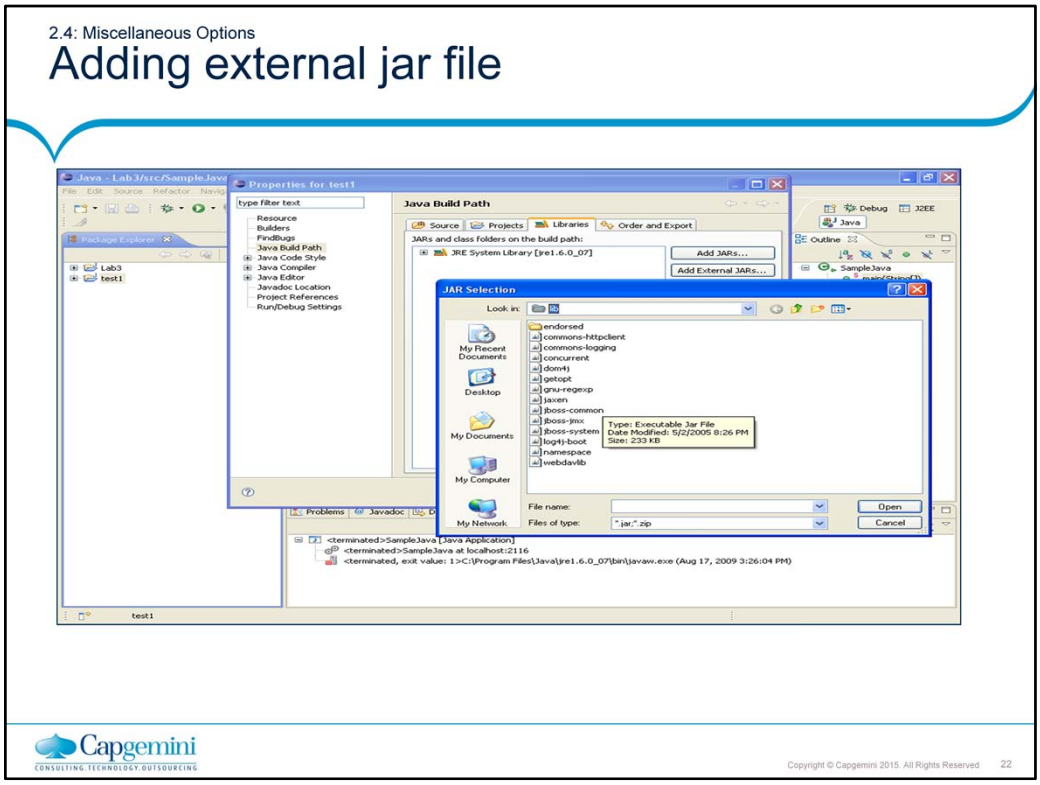
- Debugging can be attained by stepping-into or stepping-over the statements.
  - **Step-into** will traverse through each and every statement in a function.
  - **Step-over** will generate output after the function call is over.
- Tracing and watching the variable values is available as different debug views.



<u>Creating and Managing Java Projects</u>:

Debugging a Java Program:

- You may launch your Java programs from the workbench. The programs may be launched in either **run** or **debug** mode.
  - In **run** mode, the program executes. However, the execution may not be suspended or examined.
  - In **debug** mode, execution may be suspended and resumed, variables may be inspected, and expressions may be evaluated.
- Variables view gives contents of the program variables at different statements in execution.
- Breakpoints can be set for debugging, by opening the **marker-bar** pop-menu and selecting **Toggle Breakpoint**. While the Breakpoint is enabled, the thread execution suspends before the execution of the line happens. **Breakpointing** is a technique to set-up the starting point for program debugging.



Miscellaneous Options:

Adding an external jar file:

- When you are developing advanced Java programs, you might have to include some external jar files.
  - Click **Project Properties**.
  - Select **Java build Path**.
  - Click the **Libraries** tab, and click the **Add External Jar Files** button.
  - Locate the folder which contains the jar files, and click **Open**.

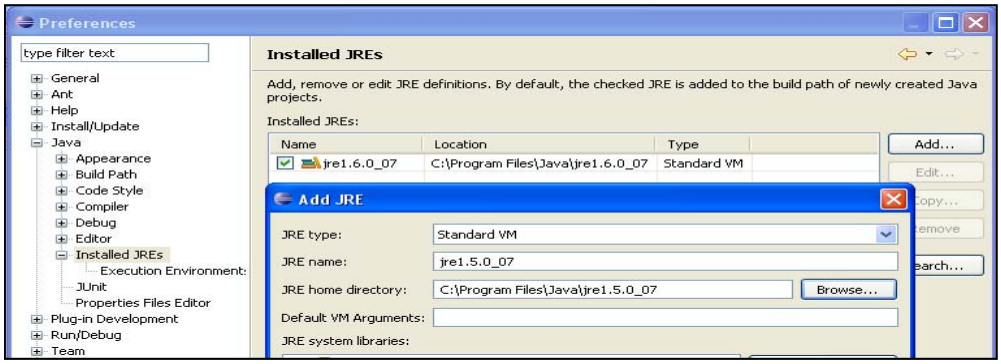
2.4: Miscellaneous Options

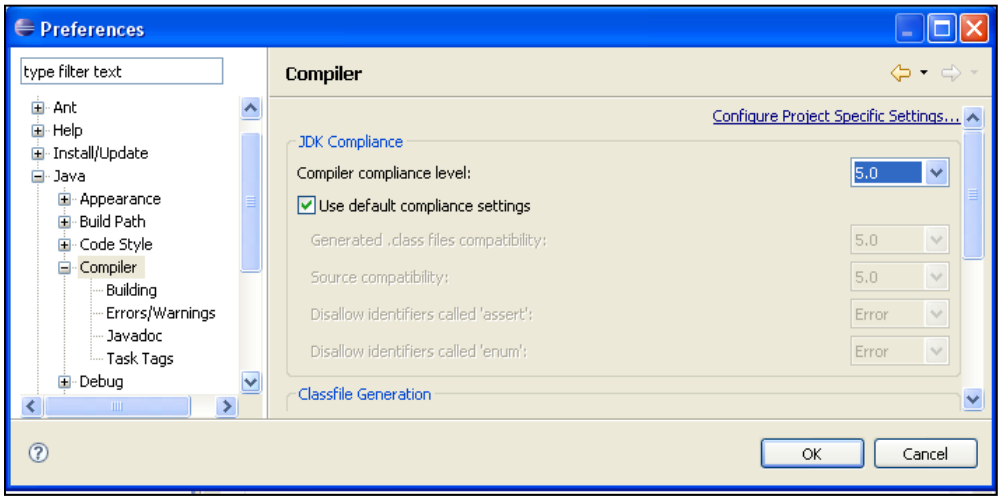
Verifying / Changing JRE Installation

- Changing the JRE is a common need while working with Eclipse which can be achieved as follows:
  - Select the menu item **Window → Preferences** to open the workbench preferences
  - Select **Java → Installed JREs** in tree pane on the left, to display the **Installed Java Runtime Environments preference** page
    - To add a new JRE, click the **Add** button, and select the new **JRE home directory**
  - Change the appropriate compiler.
    - Select **Java → Compiler** and select the appropriate compiler

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 23





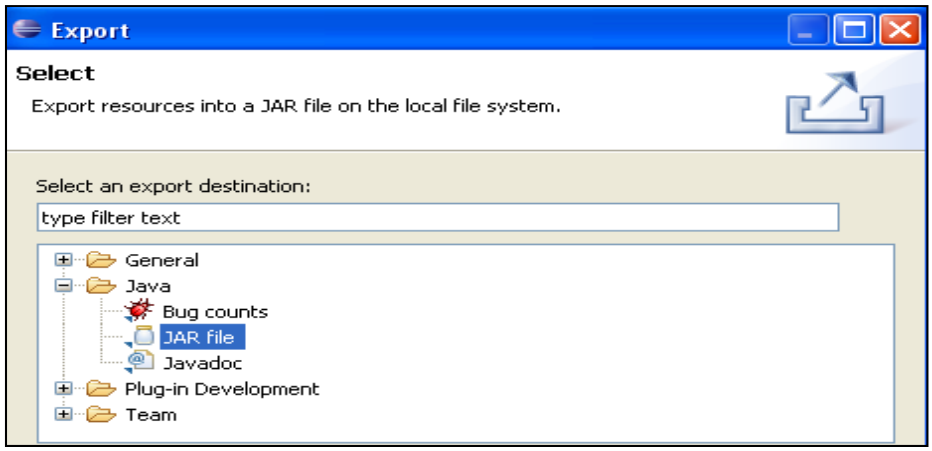
2.4: Miscellaneous Options

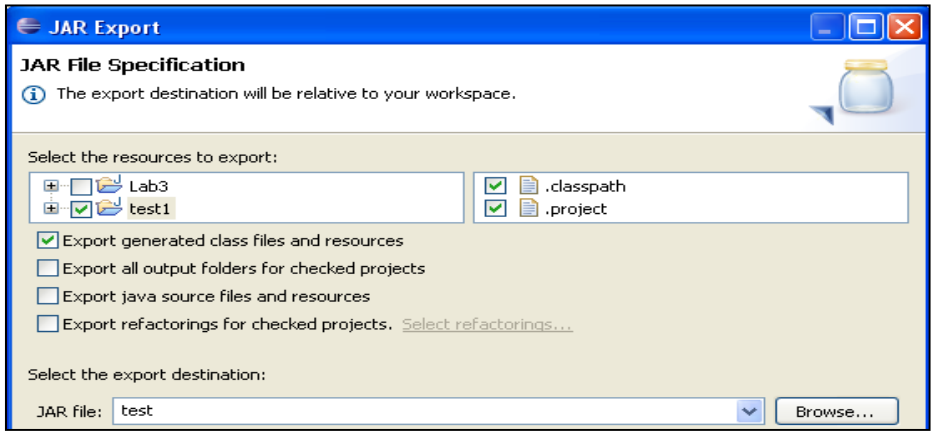
Jar File Creation

- In the Package Explorer, you can optionally pre-select one or more Java elements to export
  - Select **Export** from either the **Context** menu or from the **File** menu
  - Expand the Java node, and select **JAR** file, and click **Next**
  - On the **JAR File Specification** page, select the resources that you want to export
  - Specify a name to the JAR file
  - Click **Finish** to create the JAR file

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 24





2.4: Miscellaneous Options

Class path Setting

- Classpath variables allow you to avoid references to the location of a JAR file on your local file system
- Classpath variables can be used in a Java Build Path to avoid a reference to the local file system
- The value of such variables is configured at the following path:
  - **Window → Preferences → Java → Build Path → Classpath Variables**

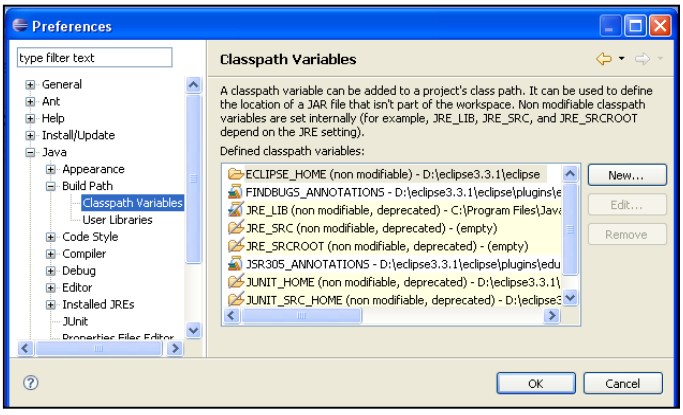
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 25

Miscellaneous Options:

Setting Classpath:

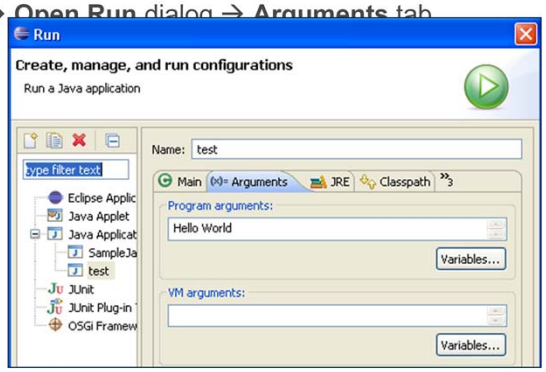
Command	Description
1. New...	It adds a new variable entry. In the resulting dialog, specify a name and path for the new variable. You can click the File or Folder buttons to browse for a path.
2. Edit...	It allows you to edit the selected variable entry. In the resulting dialog, edit the name and/or path for the variable. You can click the File or Folder buttons to browse for a path.
3. Remove…	It removes the selected variable entry.



2.4: Miscellaneous Options

Passing Command Line Arguments

- Command line arguments can be passed to the program in the following ways:
  - Select **Run** → **Open Run** dialog → **Arguments** tab



Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 26

Note: In the snapshot shown in the above slide, there are two arguments "**Hello**" and "**World**".

2.4: Miscellaneous Options

Import a Project

- To import an existing project to the workspace:
  - Go to File □ Import
  - Select Existing Projects into Workspace option
  - Select the radio button next to Select archive file, and click the Browse button
  - Find the archive file on your hard disk and click Open to select
  - If you have selected an archive file containing an entire Eclipse project, then the project name will appear in the box below, that is already checked
  - Click Finish to perform the import

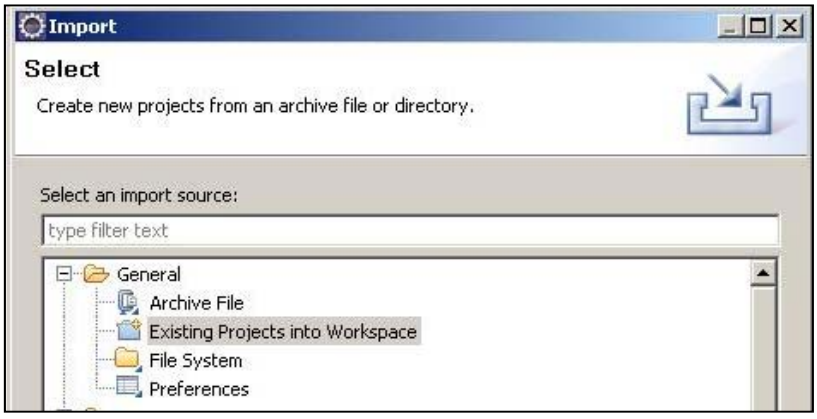
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

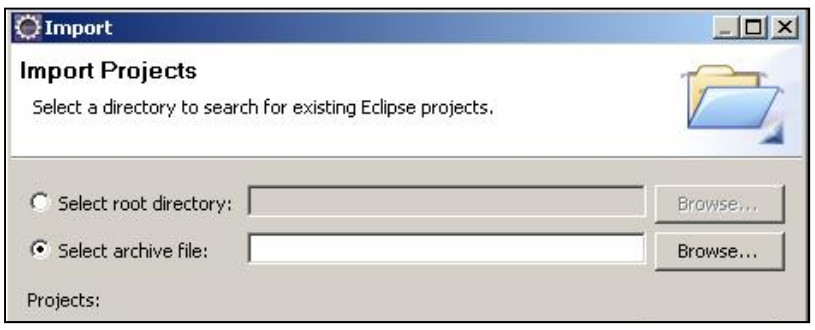
 27

Import and Export Options:

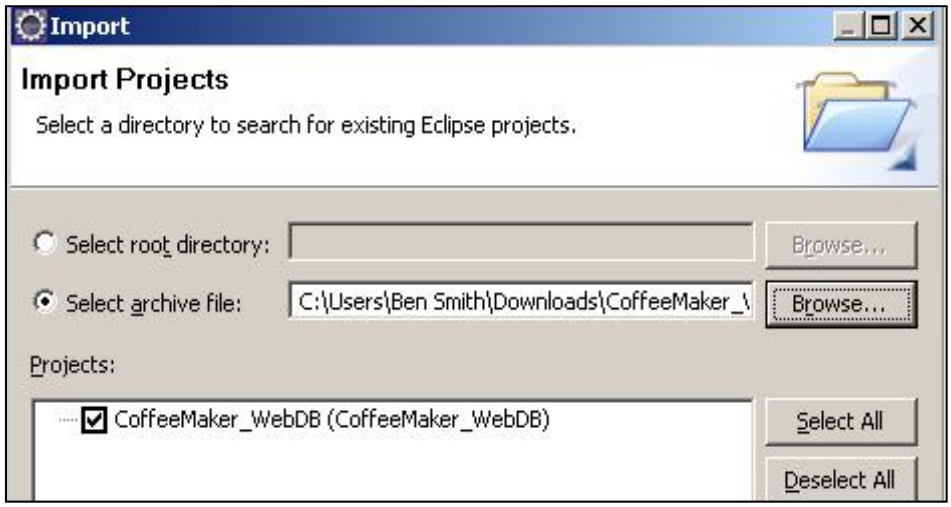
Import a Project:

The snapshots of the above steps are given below:





Import and Export Options:
Import a Project (contd.):

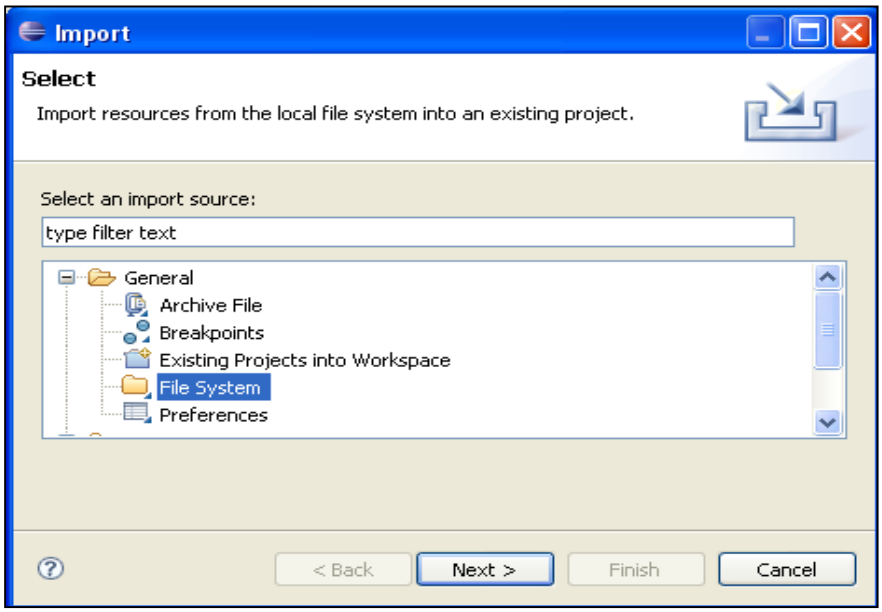


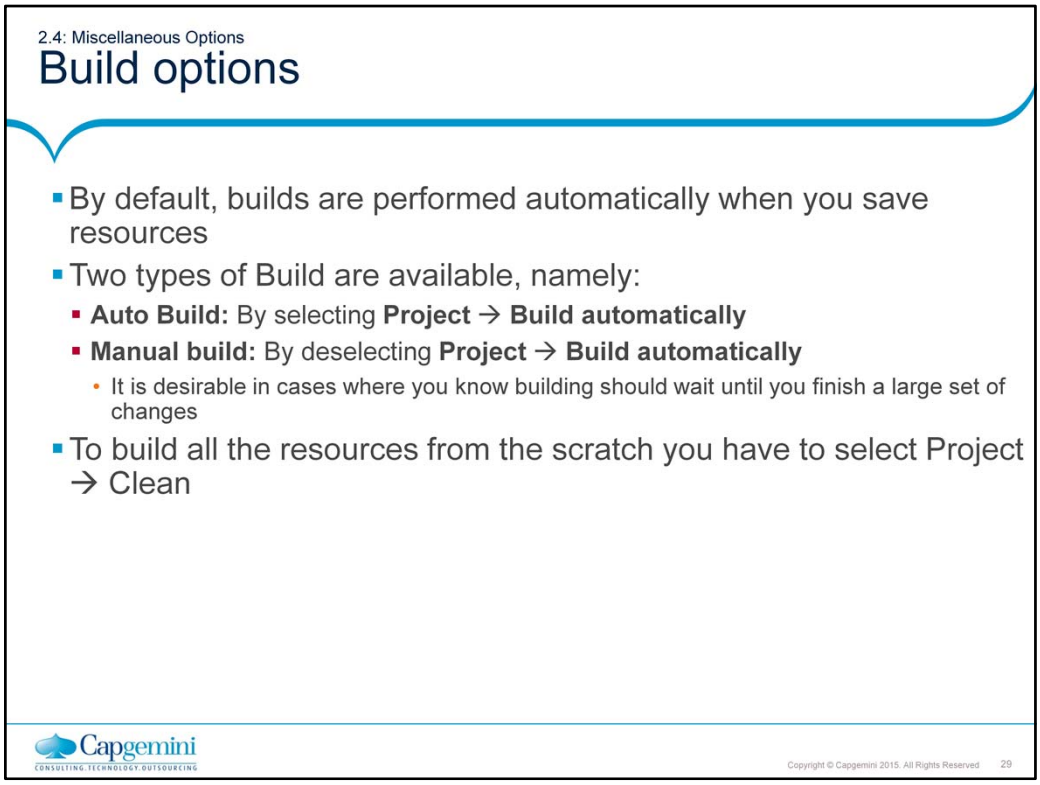
Importing Files into Project

Files can be imported into the Workbench either by:

- Dragging and dropping from the file system, or
- Copying and pasting from the file system, or
- Using the Import wizard

Using **drag and drop** or **copy and paste** to import files relies on operating system support and that is not necessarily available on all platforms. If the platform you are using does not have this support, then you can always use the **Import wizard**.





Import and Export Options:

Builds:

- Builders create or modify workspace resources, usually based on the existence and state of other resources. They are a powerful mechanism for enforcing the constraints of some domain.
  For example: A Java builder converts Java source files (.java files) into executable class files (.class files), a web link builder updates links to files whose name/location have changed, and so on.
- As resources are created and modified, builders are run and the constraints are maintained. This transform need not be one to one.
  For example: A single .java file can produce several .class files.

Import and Export Options:

Auto-build versus Manual Build:

- There are two distinct user work modes with respect to building:
  - Auto-build
  - User initiated manual build
- If you do not need fine-grained control over when builds occur, you can turn on **auto-building**. By keeping auto-building on, builds occur after every set of resource changes (for example, saving a file, importing a ZIP, ...). Auto-building is efficient because the amount of work done is proportional to the amount of change done. The benefit of auto-building is that your derived resources (for example, Java .class files) are always up to date. Auto-building is turned on/off via the **Build automatically** option accessed as **Project → Build automatically** option.
- If you need more control over when builds occur, you can turn off auto-building and invoke builds **manually**. This is sometimes desirable in cases where, for example, you know building is of no value until you finish a large set of changes. In this case, there is no benefit in paying the cost of auto-building.
- Auto-building always uses incremental building for efficiency.
- A clean build (**Project → Clean**) discards any existing built state. The next build after a clean will transform all resources according the domain rules of the configured builders.

2.4: Miscellaneous Options

General Tips and Tricks

- Creating Getters and Setters:
  - To create getter and setter methods for a field:
    - Select the field's declaration
    - Invoke Source □ Generate Getter and Setter
- Content assist:
  - Content assist provides you with a list of suggested completions for partially entered strings
  - In the Java editor, press CTRL+SPACE or invoke Edit □ Content Assist

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

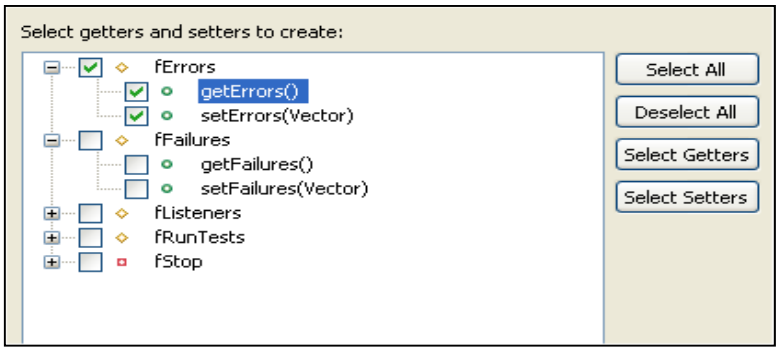
 31

Miscellaneous Options:

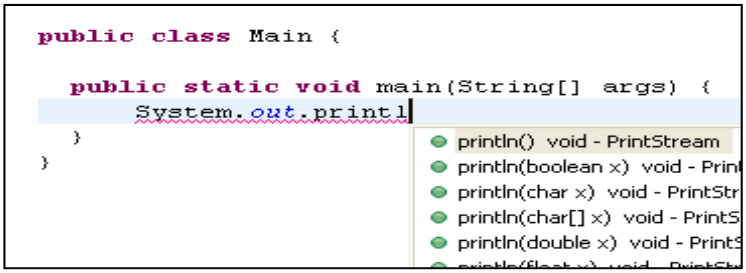
Tips and Tricks:

Creating getters and Setters :

Select the field's declaration, and invoke **Source → Generate Getter and Setter**.



Content Assist:



General Tips and Tricks

- Source menu contains a lot of options which can be used during code generation:
  - **Code Comments:** You can quickly add and remove comments in a Java expression
  - **Import Statements:** You can use it to clean up unresolved references, add import statements, and remove unneeded ones
  - **Method Stubs:** You can create a stub for an existing method by dragging it from one class to another

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

General Tips and Tricks

- **Try / Catch statements:** You can create Try / Catch block for expression by Source → Surround with try/catch
- **Javadoc Comments:** You can generate Javadoc comments for classes and methods with Source → Add Javadoc Comment
- **Superclass constructor:** Add the superclass constructors with Source → Add Constructor from Superclass

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 33

Lab: Working with Eclipse

- Lab 1.1: Working with Eclipse

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 34

Summary

- In this lesson, you have learnt:
  - The method to install Eclipse
  - Process to create a Java Project with Eclipse
  - Various useful features of Eclipse

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 35

Review Questions

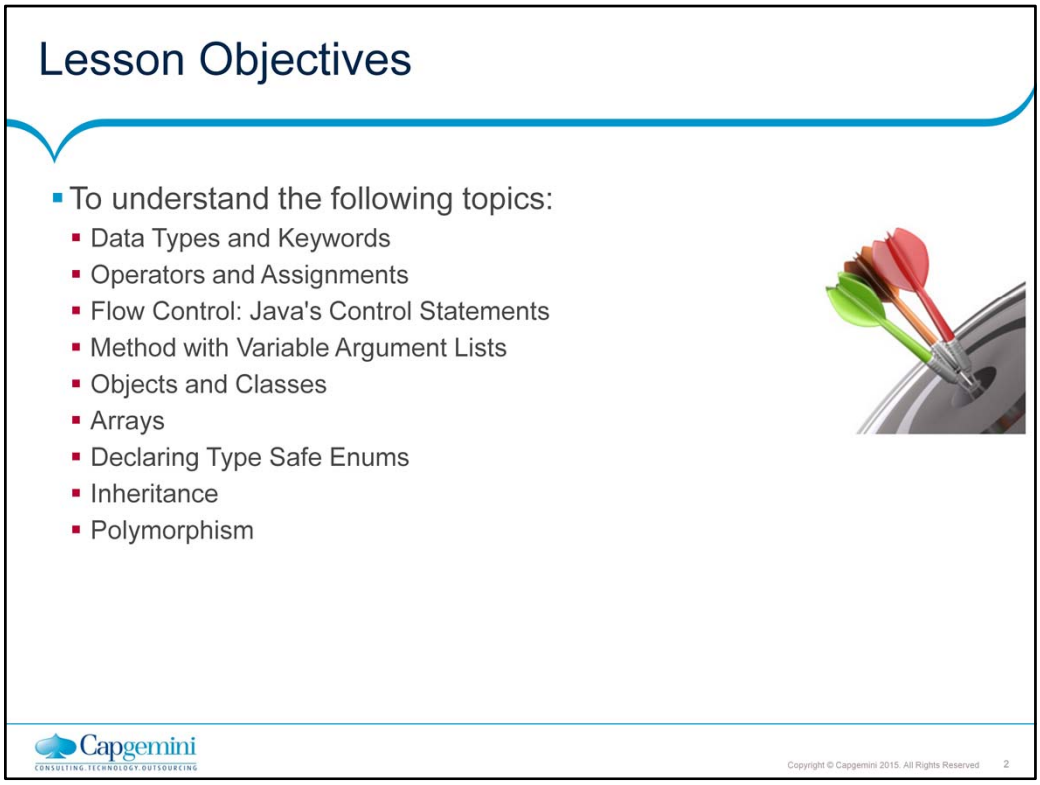
- Question 1: Which of the following are true with Eclipse 3.5?
  - **Option 1:** A Java Project in Eclipse has got a Java builder that can incrementally compile Java source files as they are changed
  - **Option 2:** A workspace can have one project only
  - **Option 3:** The source and class files can be kept in different folders

- Question 2: To build all resources, even those that have not changed since the last build, you have to select the following option:
  - **Option1:** Project → Build Project
  - **Option2:** Project → Build All
  - **Option3:** Project → Clean

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Core Java with JAXP

Lesson 3: Language Fundamentals



Lesson Objectives:

This lesson introduces to the fundamentals of the Java Programming Language.

Lesson 3: Language Fundamentals

- 3.1: Language Fundamentals
- 3.2: Data Types
- 3.3: Variables
- 3.4: Methods and Parameter Passing
- 3.5: Methods with Variable Argument Lists
- 3.6: Keywords
- 3.7: Operators and Assignments
- 3.8: Flow Control : Java's control statements
- 3.9: Objects and classes
- 3.10: Arrays
- 3.11: Declaring Type Safe Enums
- 3.12: OOPS features in Java like Inheritance, Polymorphism
- 3.13: Other modifiers like Abstract, Static and Final

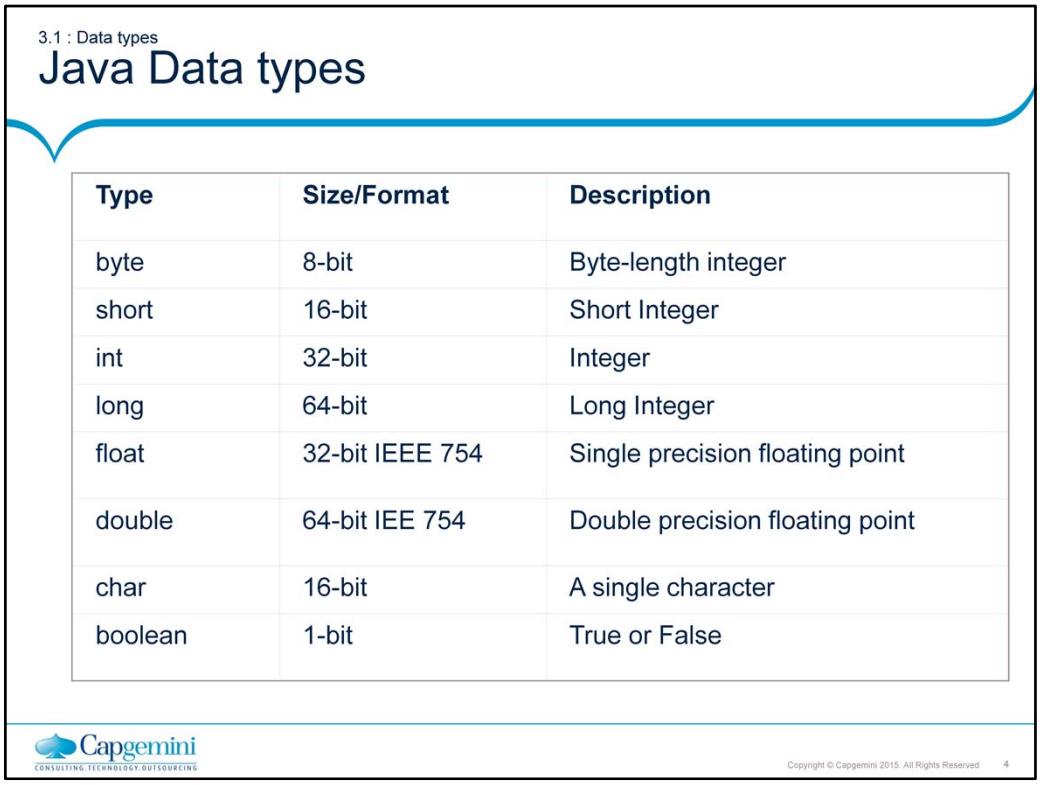
Lesson Objectives (contd..)

- Other Modifiers – abstract, final, static
- Object Class
- System Class
- String Handling
- Wrapper Classes
- Common Best Practices

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 3

3.15: The Object Class
3.16: The System Class
3.17: String Handling
3.18: Wrapper Classes
3.19: Simple formatted I/O and Scanner
3.20: Common Best Practices



There are two data types available in Java:
Primitive Data Types.
Reference/Object Data Types :- is discussed later.

There are eight primitive data types supported by Java (see slide above).
Primitive data types are predefined by the language and named by a key word.

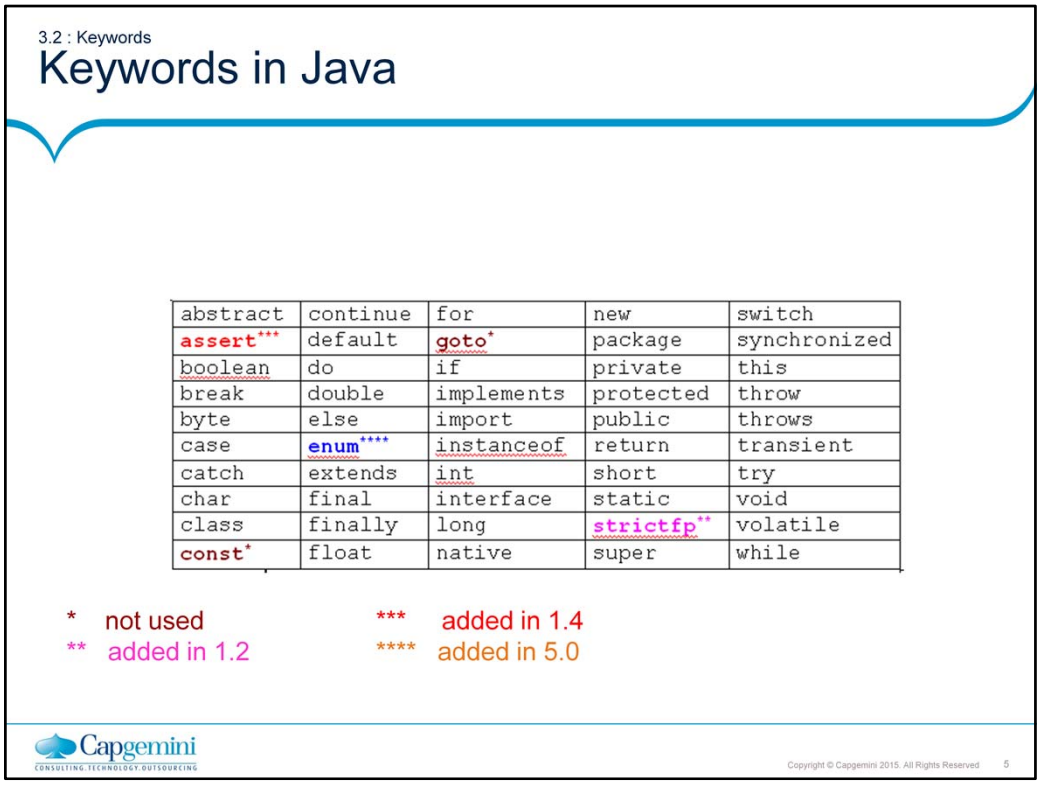
The default character set used by Java language is **Unicode character** set and hence a **character data type** will consume **two bytes** of memory instead of a byte (a standard for ASCII character set).
Unicode is a character coding system designed to support text written in diverse human languages.

This allows you to use characters in your Java programs from various alphabets such as Japanese, Greek, Russian, Hebrew, and so on.
This feature **supports** a readymade support for **internalization** of java.

The default values for the various data types are as follows:

Integer : 0
Character : '\u0000'
Decimal : 0.0
Boolean : false
Object Reference: null

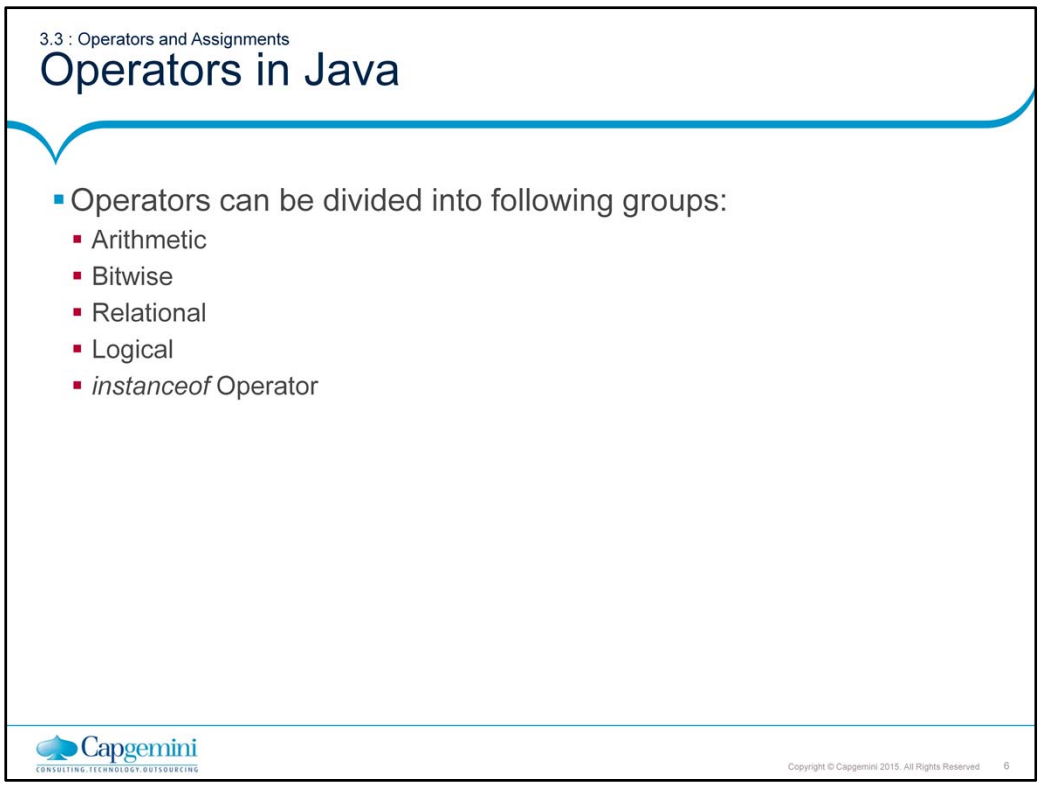
[Note: C or C++ data types pointer, struct, and union are not supported. Java does not have a typedef statement (as in C and C++).]



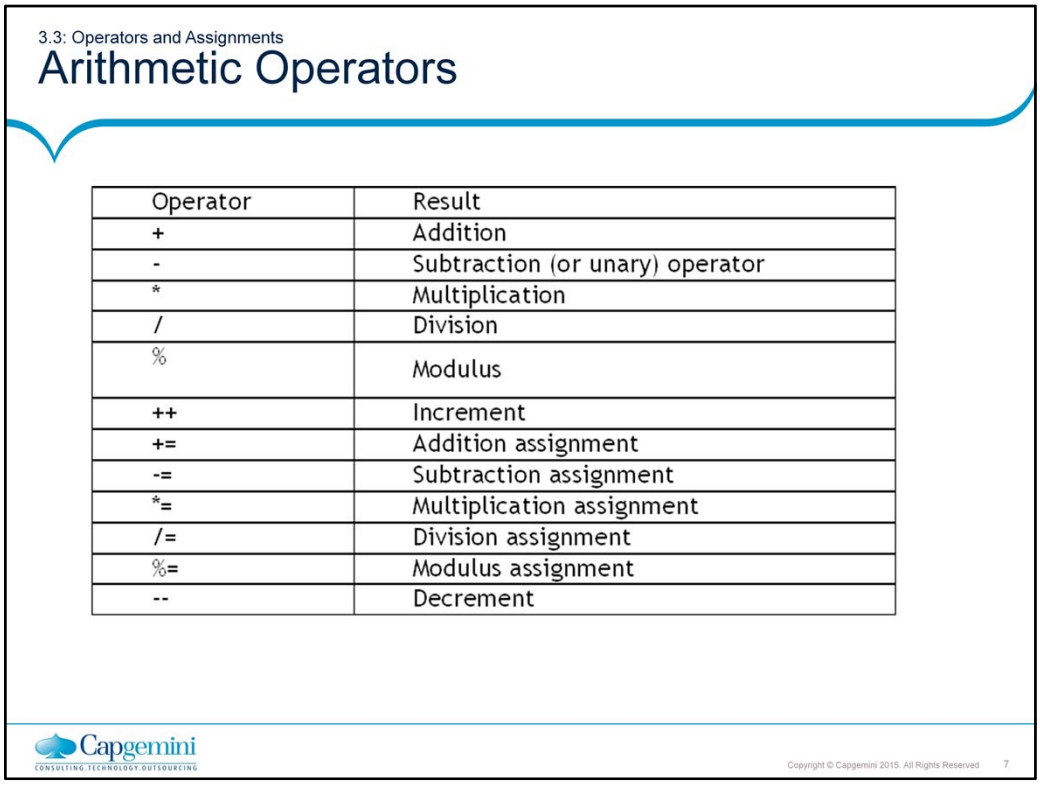
Keywords are reserved identifiers that are predefined in the language and cannot be used to denote other entities. E.g. class, boolean, abstract, do, try etc. Incorrect usage results in compilation errors.

In addition, three identifiers are reserved as predefined literals in the language: null, true and false.

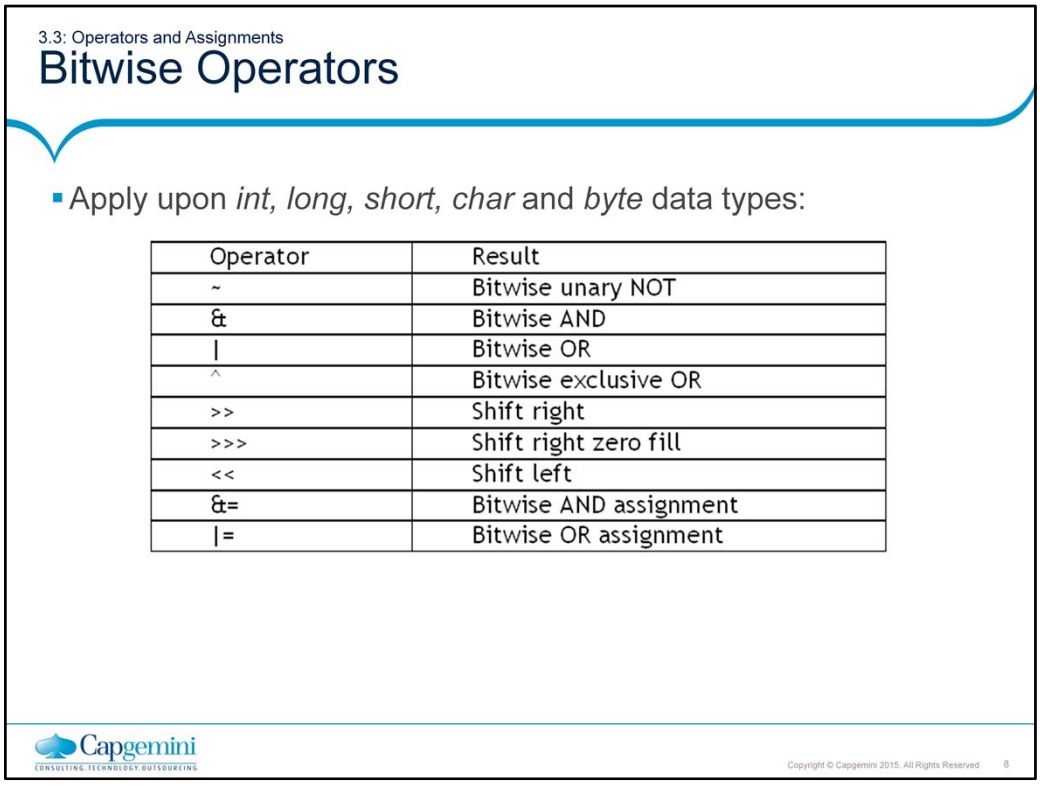
The table above shows the keywords available in Java 5.



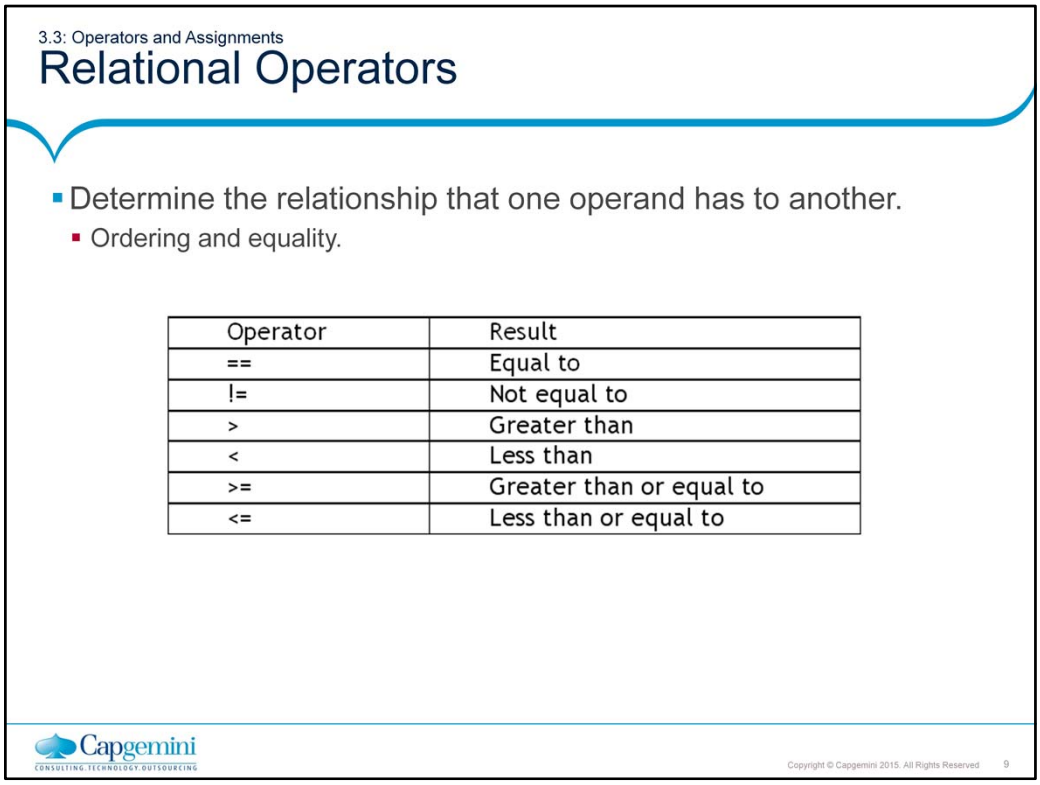
Java provides a rich set of operators to manipulate variables. These are classified into several groups as shown above.



Arithmetic operators are summarized in the table above:
Integer division yields an integer quotient for example, the expression 7 / 4 evaluates to 1, and the expression 17 / 5 evaluates to 3. Any fractional part in integer division is simply discarded (i.e., truncated) no rounding occurs.
Java provides the remainder operator, %, which yields the remainder after division. The expression x % y yields the remainder after x is divided by y. Thus, 7 % 4 yields 3, and 17 % 5 yields 2. This operator is most commonly used with integer operands, but can also be used with other arithmetic types.
Parentheses are used to group terms in Java expressions in the same manner as in algebraic expressions. For example, to multiply a times the quantity b + c, we write **a*(b+c).**
If an expression contains nested parentheses, such as **((a+b)*c)** the expression in the innermost set of parentheses (a + b in this case) is evaluated first.
Order of Precedence:
Multiplication, division and remainder operations are applied first. If an expression contains several such operations, the operators are applied from left to right. **Multiplication, division and remainder operators have the same level of precedence**.
Addition and subtraction operations are applied next. If an expression contains several such operations, the operators are applied from **left to right**. **Addition and subtraction operators have the same level of precedence.**

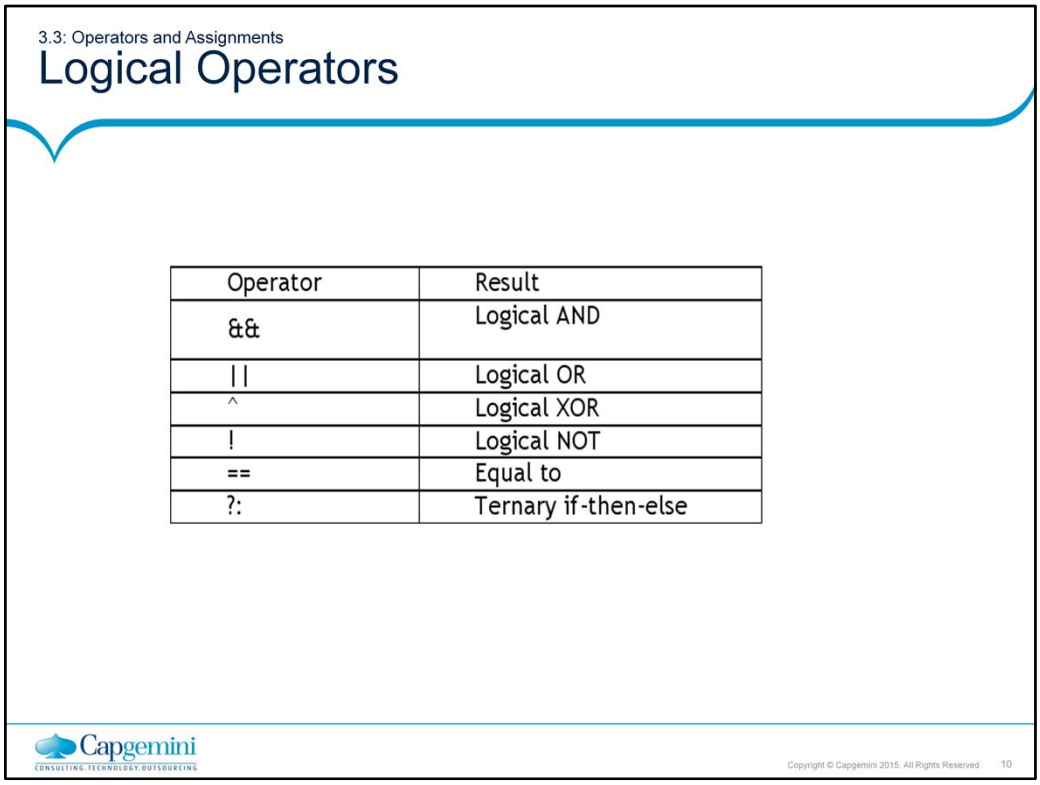


The Java programming language also provides operators that perform bitwise and bit shift operations on integral types. The operators discussed in this section are less commonly used.
The unary bitwise complement operator "~" inverts a bit pattern; it can be applied to any of the integral types, making every "0" a "1" and every "1" a "0". For example, a byte contains 8 bits; applying this operator to a value whose bit pattern is "00000000" would change its pattern to "11111111".
The signed left shift operator "<<" shifts a bit pattern to the left, and the signed right shift operator ">>" shifts a bit pattern to the right. The bit pattern is given by the left-hand operand, and the number of positions to shift by the right-hand operand. The unsigned right shift operator ">>>" shifts a zero into the leftmost position, while the leftmost position after ">>" depends on sign extension.
Bitwise & operator performs a bitwise AND operation.
Bitwise ^ operator performs a bitwise exclusive OR operation.
Bitwise | operator performs a bitwise inclusive OR operation.



A condition is an expression that can be either true or false. For example, the condition "grade is greater than or equal to 60" determines whether a student passed a test. If the condition in an if statement is true, the body of the if statement executes. If the condition is false, the body does not execute.

Conditions in if statements can be formed by using the equality operators (== and !=) and relational operators (>, <, >= and <=). Both equality operators have the same level of precedence, which is lower than that of the relational operators. The equality operators associate from left to right. The relational operators all have the same level of precedence and also associate from left to right.



Java provides logical operators to enable programmers to form more complex conditions by combining simple conditions. The logical operators are && (conditional AND), || (conditional OR), & (boolean logical AND), | (boolean logical inclusive OR), ^ (boolean logical exclusive OR) and ! (logical NOT).

3.3: Operators and Assignments

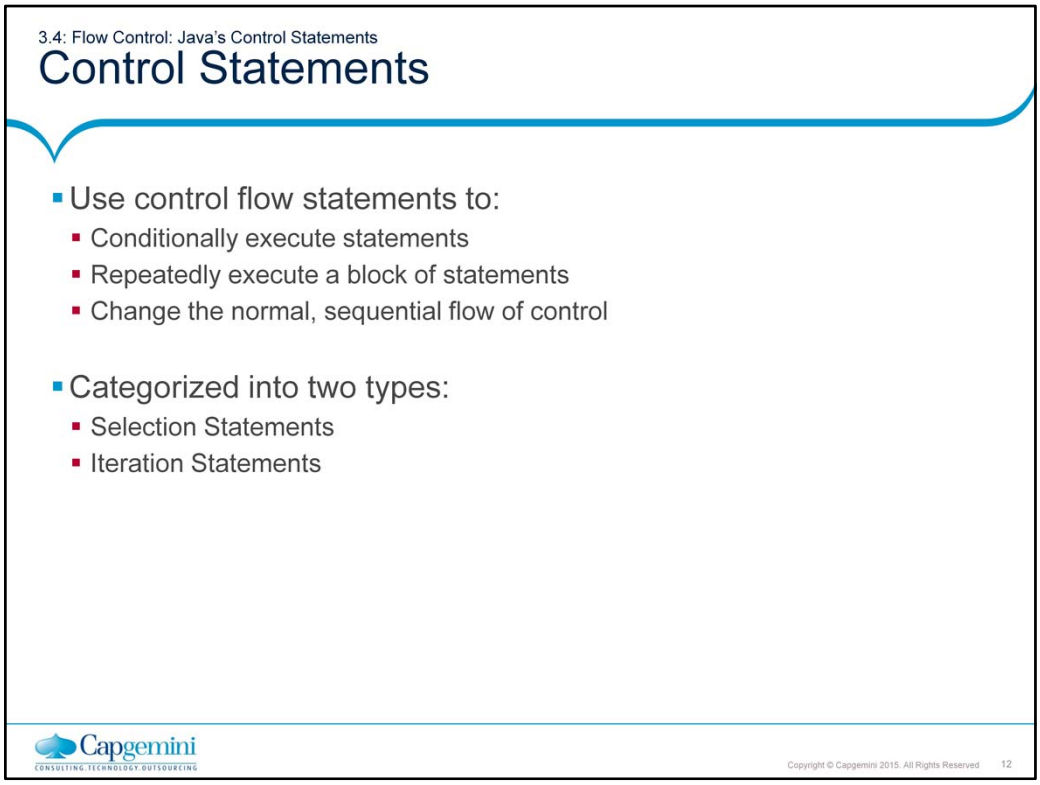
instanceOf Operators

- The *instanceof* operator compares an object to a specified type
- Checks whether an object is:
  - An instance of a class.
  - An instance of a subclass.
  - An instance of a class that implements a particular interface.
  - Example : The following returns *true*:

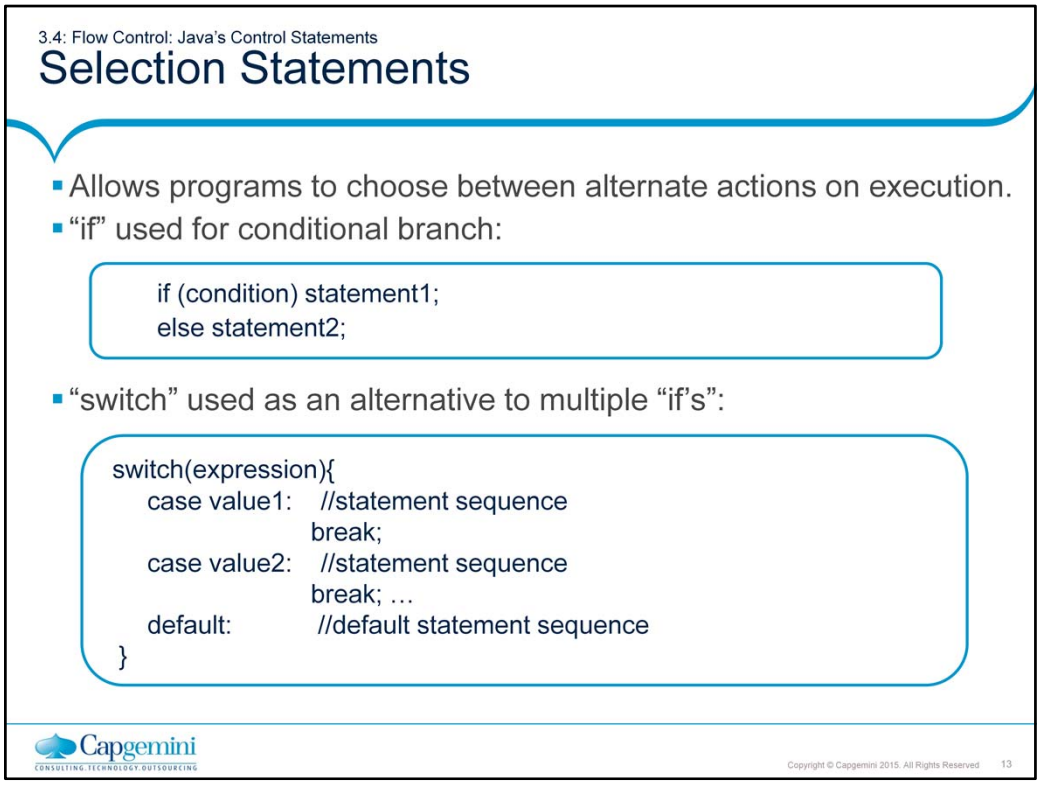
```
new String("Hello") instanceof String;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 11



Java being a programming language, offers a number of programming constructs for decision making and looping.



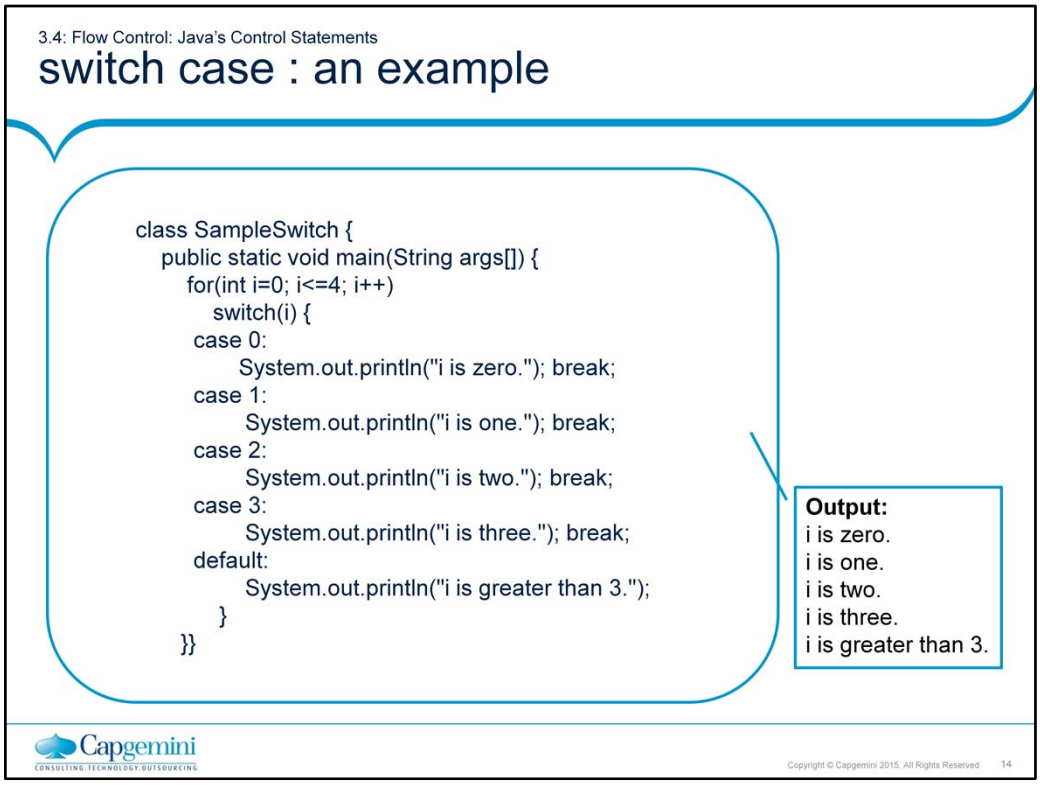
If Statement:

The if statement is Java's conditional branch statement. It can be used to route program execution through two different paths.

Each statement may be a single statement or a compound statement enclosed in curly braces. The condition is any expression that returns a boolean value. The else clause is optional.

The if works like this: If the condition is true, then statement1 is executed. Otherwise, statement2 (if it exists) is executed. In no case will both statements be executed. For example, consider the following:

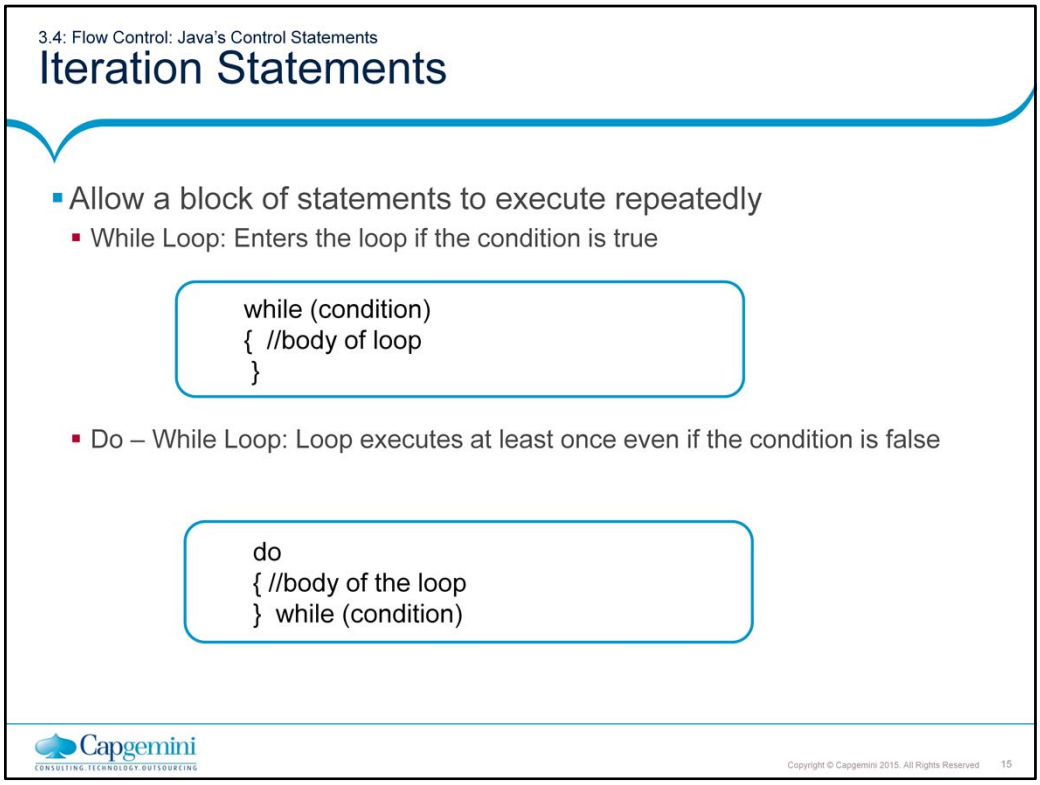
```
int a, b;
if(a < b) a = 0;
else b = 0;
```



Switch – Case:

The *switch* statement is Java's multi-way branch statement. It provides an easy way to dispatch execution to different parts of your code based on the value of an expression.

As such, it often provides a better alternative than a large series of *if-else-if* statements.



while statement:

The body of the loop is executed as long as the conditional expression is true. When *condition* becomes false, control passes to the next line of code immediately following the loop. Example:

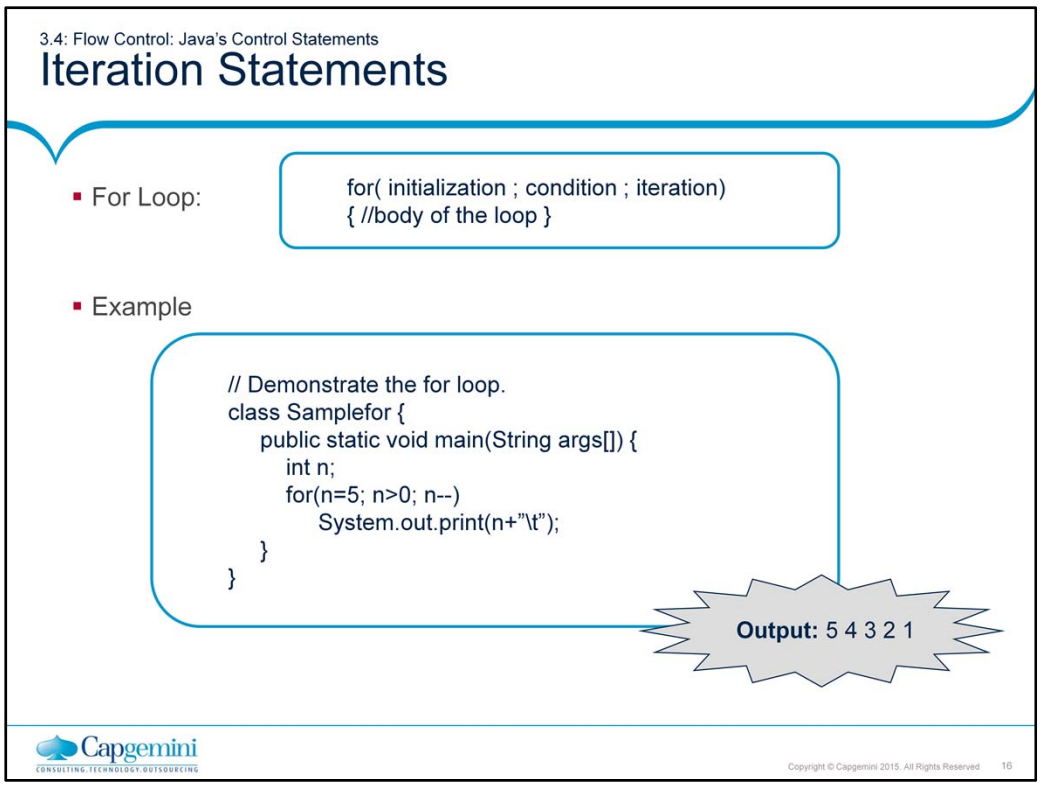
```
class Samplewhile {
        public static void main(String args[]) {
            int n = 5;
            while(n > 0) {
               System.out.print(n+"\t");
                n--;
    } } }
```

Output: 5 4 3 2 1

The while loop evaluates its conditional expression at the top of the loop. Hence, if the condition is false to begin with, the body of the loop will not execute even once.

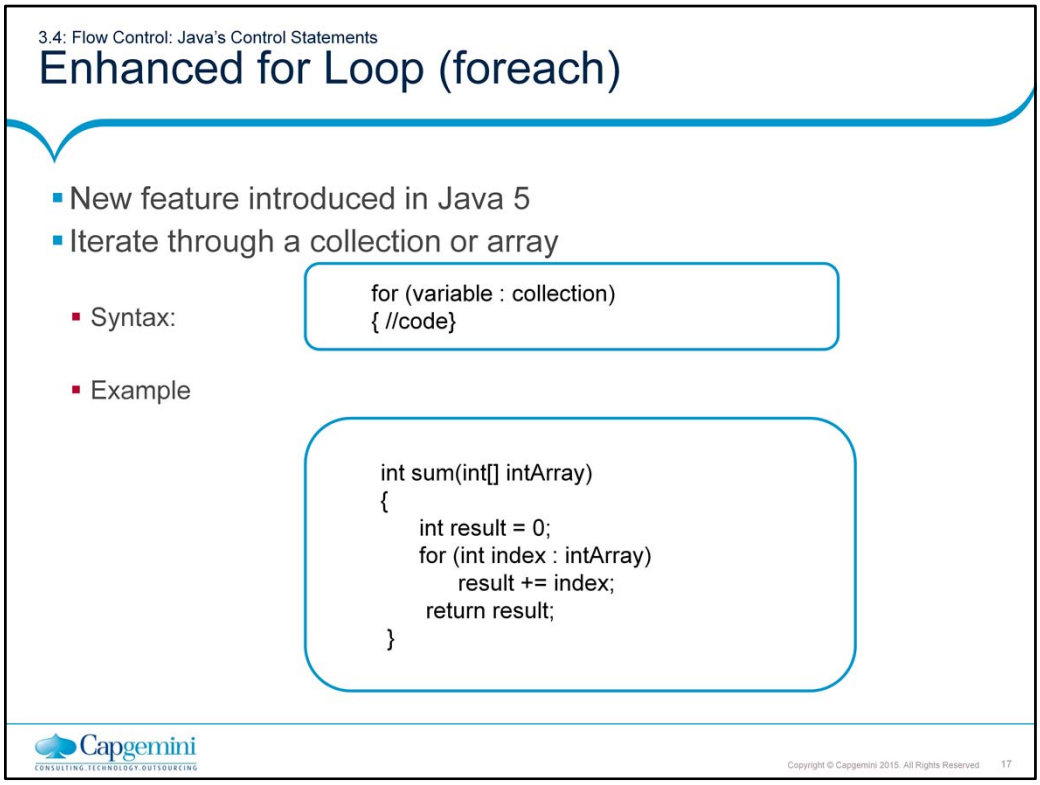
do – while loop:

This construct executes the body of a **while** loop at least once, even if the conditional expression is false to begin with. The termination expression is tested at the **end** of the loop rather than at the beginning.



for loop:

When the **for** loop first starts, the *initialization* portion of the loop is executed. Generally, this is an expression that sets the value of the *loop control variable,* which acts as a counter that controls the loop. The initialization expression is only executed once. Next, *condition* is evaluated. It usually tests the loop control variable against a target value. If this expression is true, then the body of the loop is executed, else the loop terminates. Next, the *incremental* portion of the loop is executed. This is usually an expression that increments or decrements the loop control variable. The loop then iterates, first evaluating the conditional expression, then executing the body of the loop, and then executing the iteration expression with each pass. This process repeats until the controlling expression is false.



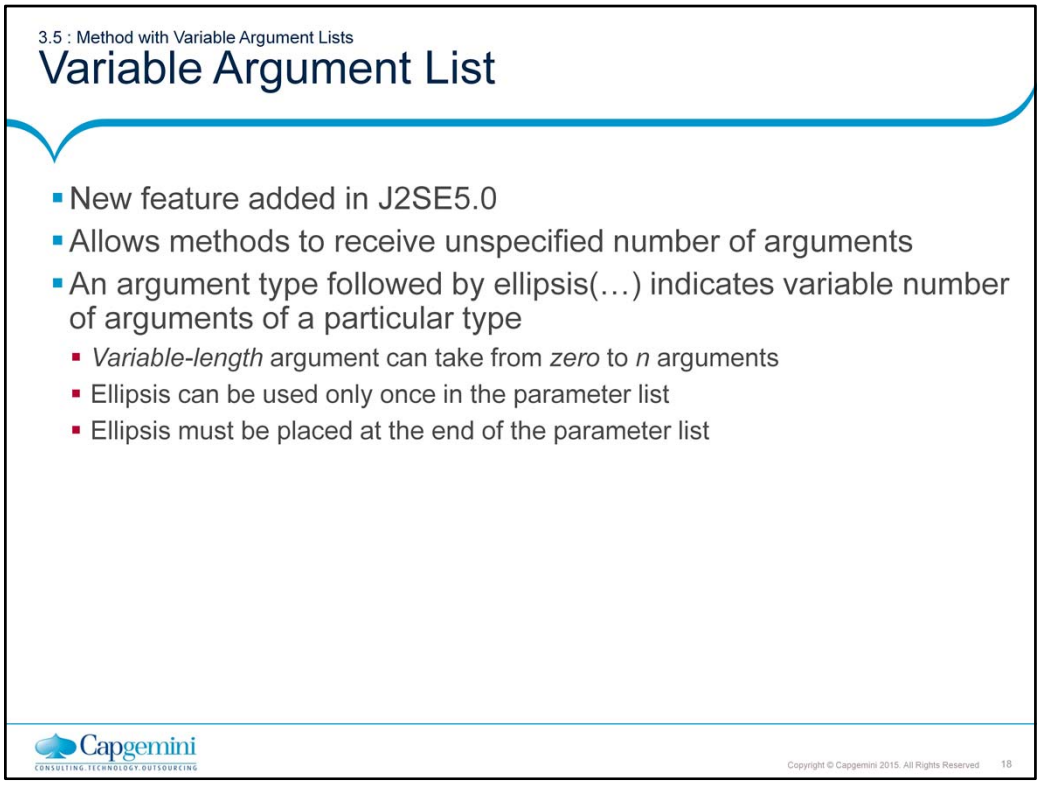
Enhanced for loop works with collections and arrays. Notice the difference between the old code where the three standard steps of initialization, conditional check and re-initialization are explicitly required to be mentioned.

Old code

```
public class OldForArray {
  public static void main(String[] args){
     int[] squares = {0,1,4,9,16,25};
     for (int i=0; i< squares.length; i++){
          System.out.printf("%d squared is %d.\n",i, squares[i]);
      }
  }}
```

New Code

```
public class NewForArray {
   public static void main(String[] args) {
       int j = 0;
       int[] squares = {0, 1, 4, 9, 16, 25};
       for (int i : squares) {
           System.out.printf("%d squared is %d.\n", j++, i);
       }
    }
}
```



J2SE5 has added a new feature that simplifies the creation of methods that need to take a variable number of arguments. This feature is called *varargs* and it is short for variable-length arguments.

3.5 : Method with Variable Argument Lists

Variable Argument List (contd..)

```
//Valid Code
void print(int a,int b,String...c)
{
        //code
}
```

```
//Invalid  Code
void print(int a, int b…,float c)
{
        //code
}
```

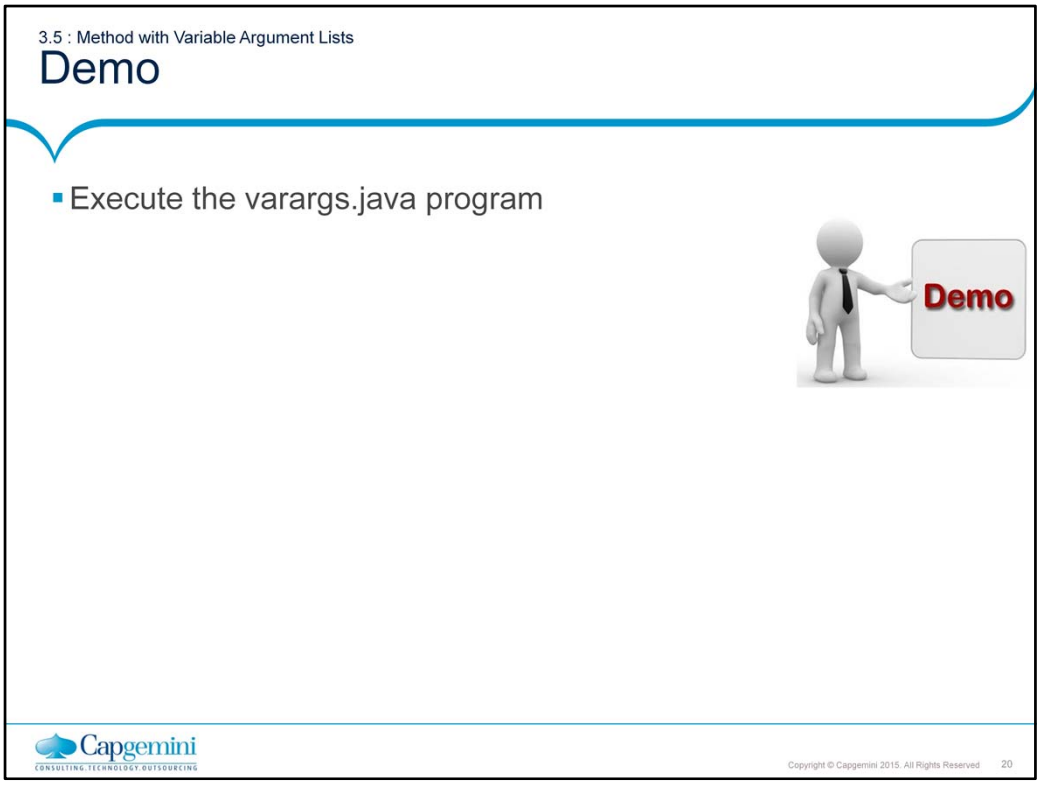
- The above print function can be invoked using any of the invocations:
  - print(1,1,"XYZ")
  - print(2,5)
  - print(5,6,"A","B")

Varargs can be used only in the final argument position.

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

 19

Note that the ellipses (…) must come only after the last parameter. Putting it anywhere else is invalid.
The three periods indicate that the final argument may be passed as an **array** *or* as a sequence of arguments.

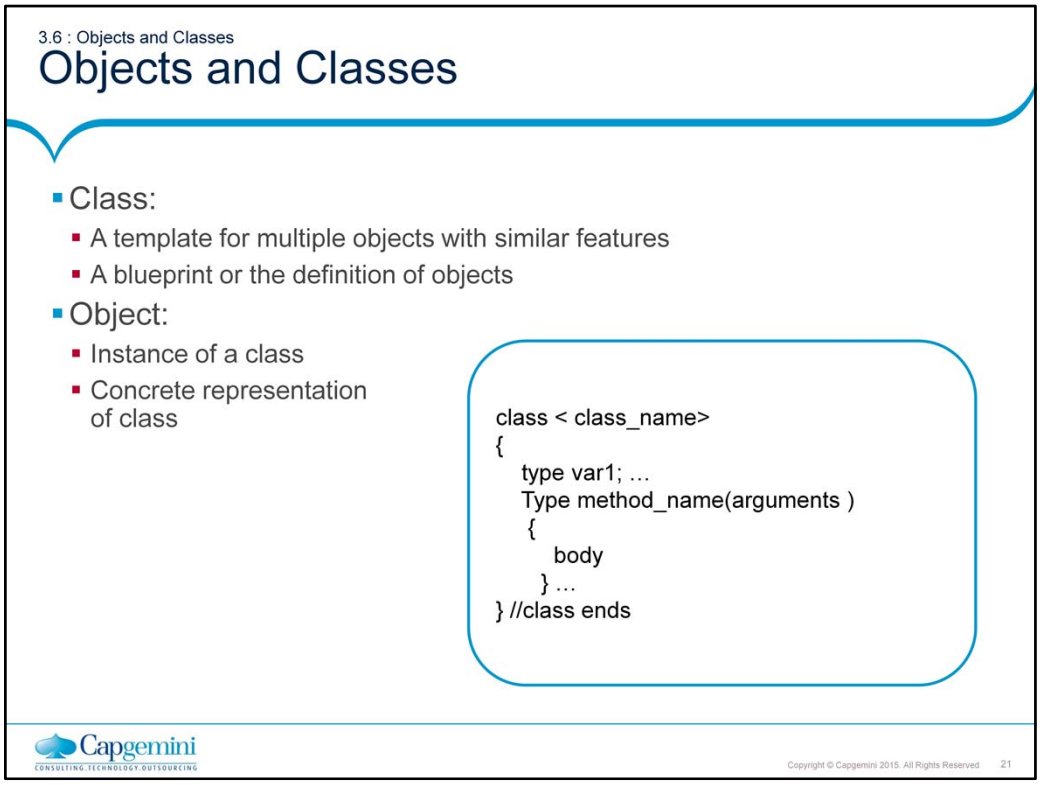



```
import static java.lang.System.*;
        public class varargs {
            static void print(int a,int y,String...s)  {
                out.println(a+""+y);
                for(int i=0;i<s.length;i++) out.print(s[i]+"\t");
                    out.println();
             }
             public static void main(String[] arg)  {
                print(3,2,"java","java 5");
                out.println("Next invoke");
                print(1,2,"a","b","c","d","e");
             } }
```

Java Arrays expose a property called length that returns the length of the array.

O/P:

32
java java 5
Next invoke
12
a b c d e



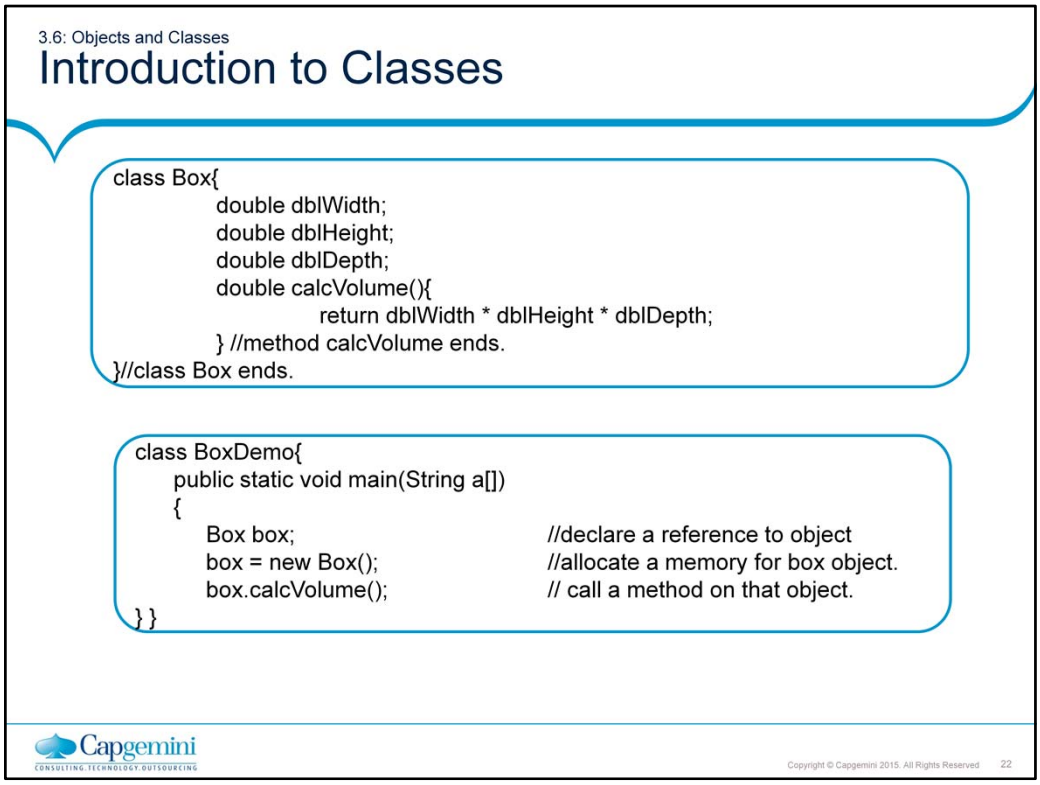
Classes describe objects that share characteristics, methods, relationships, and semantics. Each class has a **name**, **attributes** (its values determine state of an object), and **operations** (which provides the behavior for the object).

What is the relationship between objects and classes? What exists in real world is objects. When we classify these objects on the basis of commonality of structure and behavior, the result that we get are the classes.

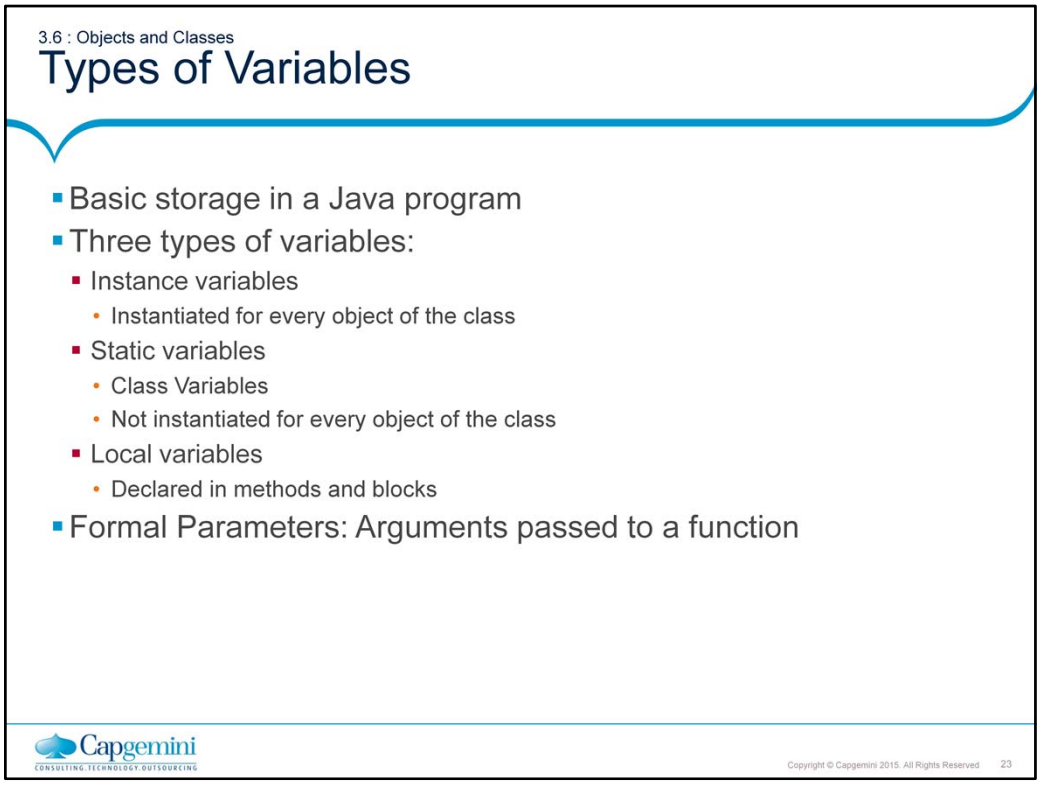
Classes are “logical”. They don’t really exist in real world. When writing software programs, it is the classes that get defined first. These classes serve as a blueprint from which objects are created.

Reference: Refer to OOP material for a detailed discussion on Object-Oriented Programming Concept.

Note: Java does not have functions defined outside classes (as C++ does).

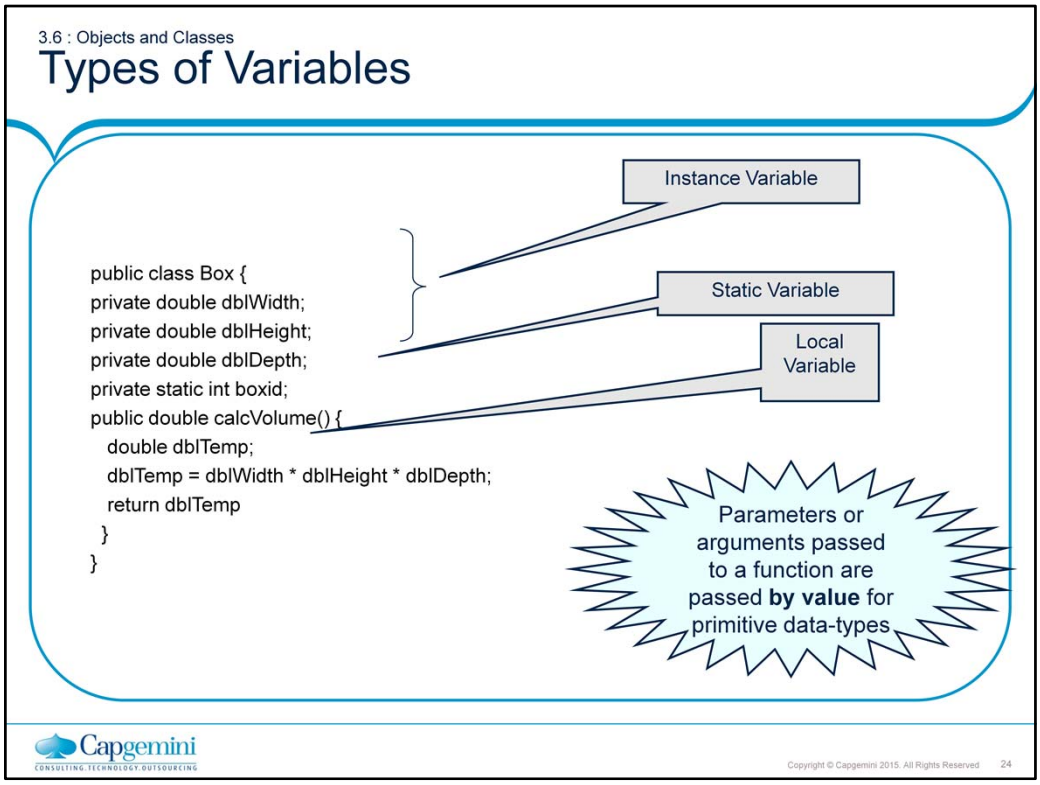


This method returns 0.0, since none of the instance members have been set. But the code shows how objects are instantiated and how methods are invoked.



Instance variables: These are members of a class and are instantiated for every object of the class. The values of these variables at any instant constitute the *state* of the object.
Static variables: These are also members of a class, but these are not instantiated for any object of the class and therefore belong only to the class. We shall be covering the static modifier in later section.
Local variables: These are declared in methods and in blocks. They are instantiated for every invocation of the method or block. In Java, local variables must be declared before they are used.
Life-cycle of the variable is controlled by the scope in which those are defined.
Formal parameters: These are the arguments passed to a function, which are similar to local variables.

Refer the example on the subsequent slide.

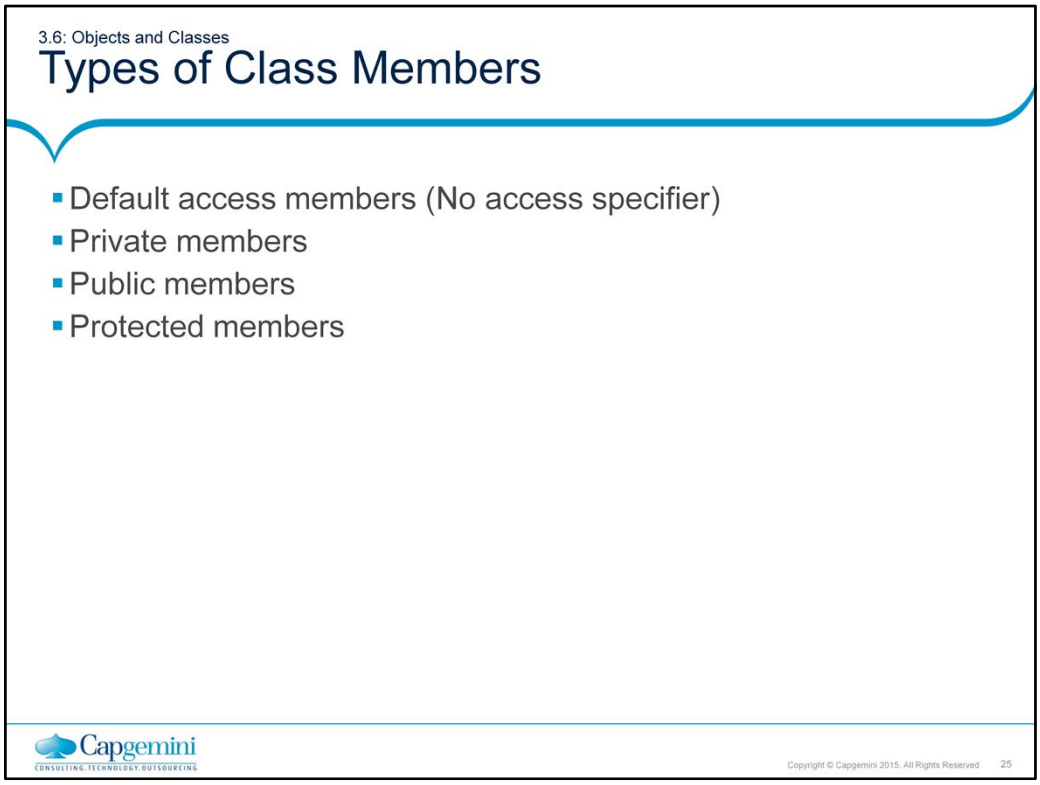


Methods and Parameter Passing:

Parameters or arguments passed to a function are passed by value for primitive data-types.

Parameters or arguments passed to a function are passed by reference for non-primitive data-types

Example: All Java objects.



public access modifier
Fields, methods and constructors declared public (least restrictive) within a public class are visible to any class in the Java program, whether these classes are in the same package or in another package.
private access modifier
Fields, methods or constructors declared private (most restrictive) cannot be accessed outside an enclosing class. This modifier cannot be used for classes. It also cannot be used for fields and methods within an interface. A standard design strategy is to make all fields private and provide public getter methods for them.
protected access modifier
Fields, methods and constructors declared protected in a superclass can be accessed only by subclasses in other packages. Classes in the same package can also access protected fields, methods and constructors, even if they are not a subclass of the protected member's class. This modifier cannot be used for classes. It also cannot be used for fields and methods within an interface.
default access modifier
Default specifier is used when no access modifier is present. Any class, field, method or constructor that has no declared access modifier is accessible only by classes in the same package. The default modifier is not used for fields and methods within an interface.

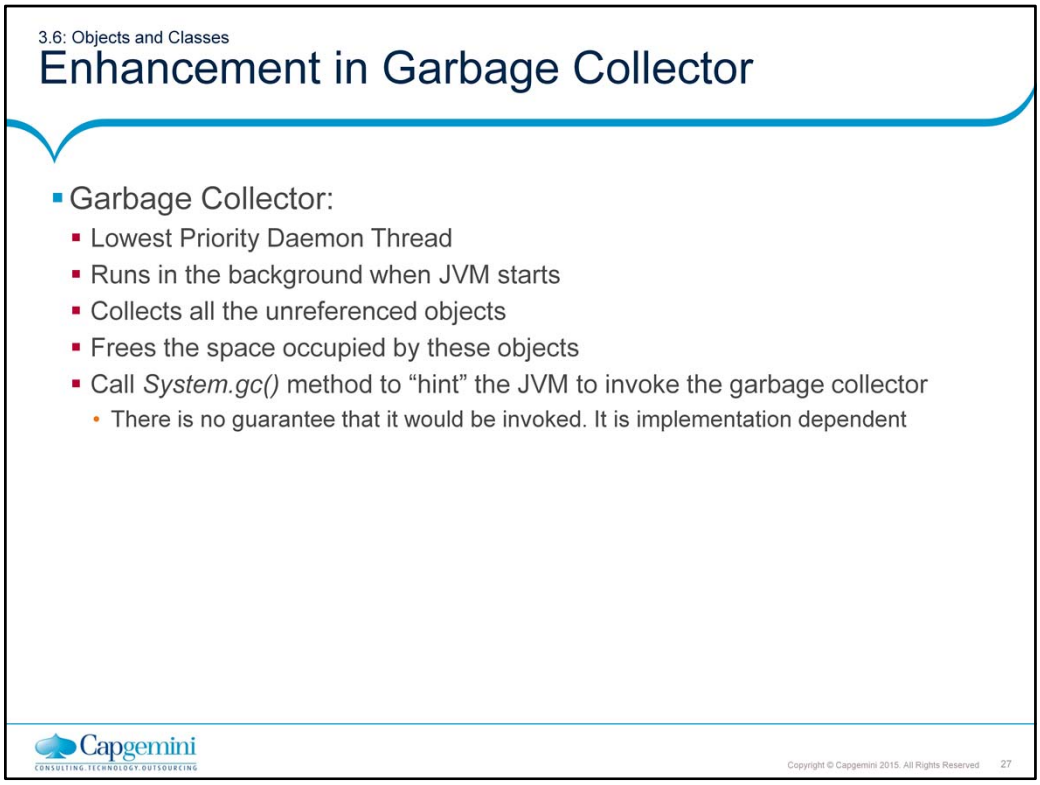
3.6: Objects and Classes

Memory Management

- Dynamic and Automatic
- No *Delete* operator
- Java Virtual Machine (JVM) de-allocates memory allocated to unreferenced objects during the garbage collection process

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

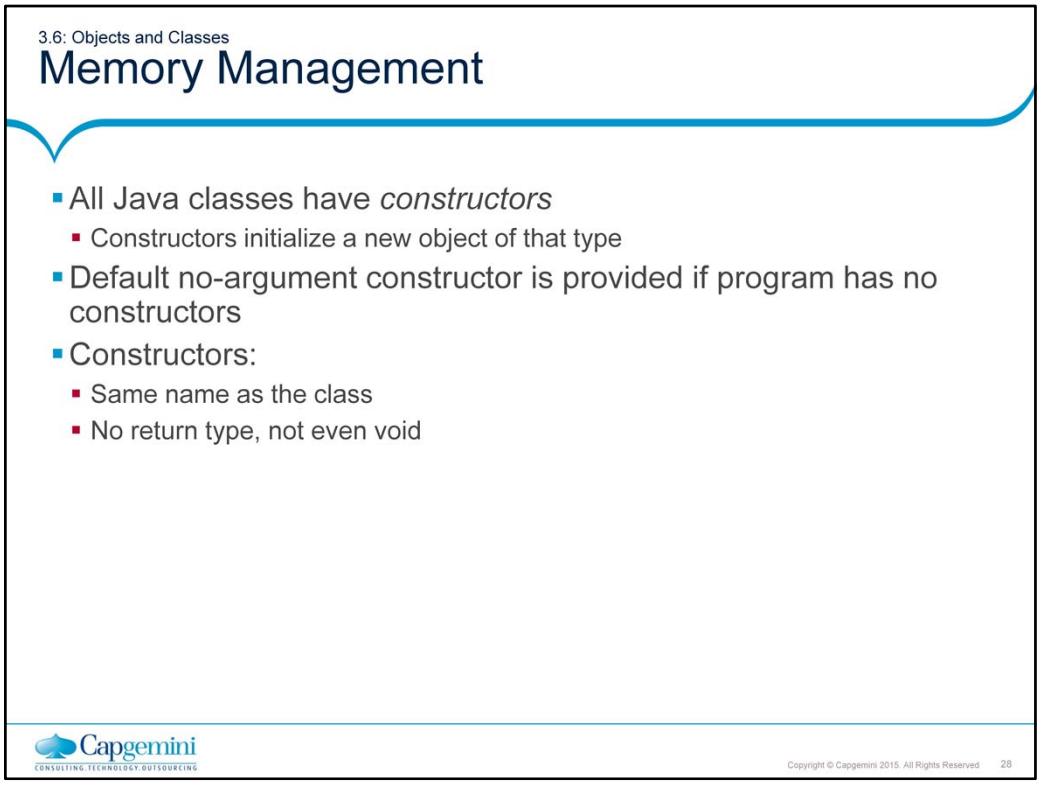
 26



There is a common misconception that *system.gc()* invokes the garbage collector, however that is not true. It just gives a request or hint to JVM to start garbage collector, but JVM may not start it immediately or even till end of the program execution. It is JVM implementation dependent issue, as to when it would start. It can even do some optimization by starting garbage collection, only when certain amount of memory is consumed etc.

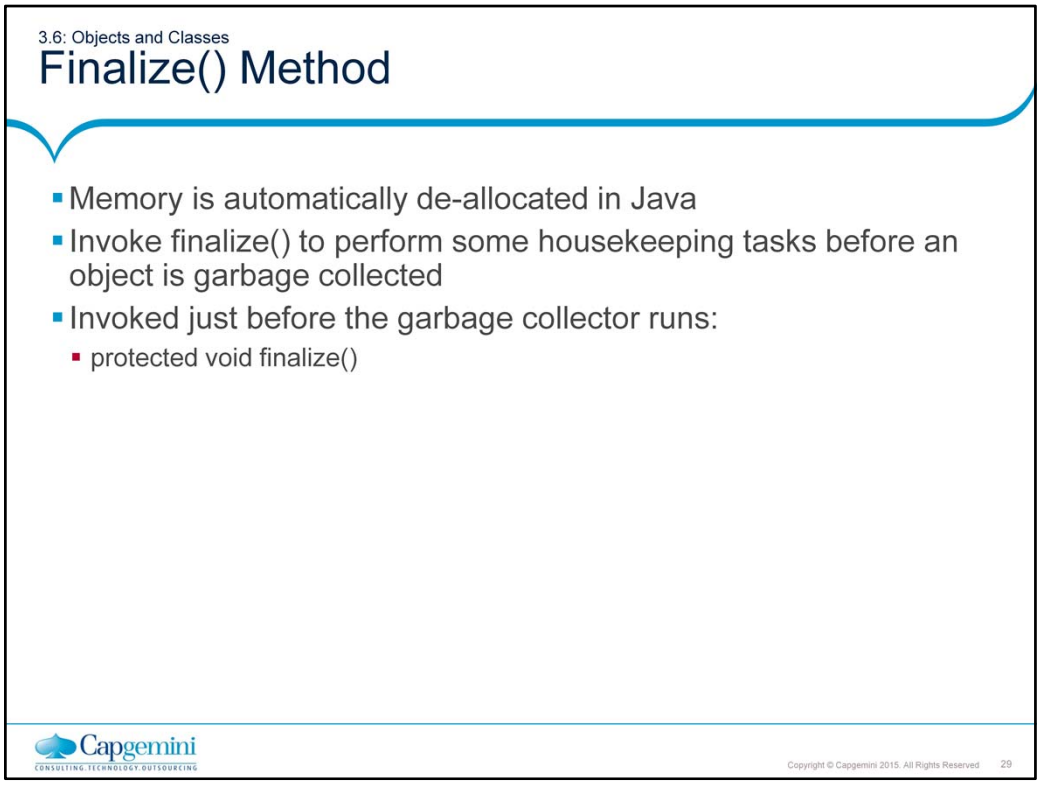
An object is eligible for garbage collection when there are no more references to that object. References that are held in a variable are naturally dropped when the variable goes out of scope. So, you can explicitly drop an object reference by setting the value of a variable whose data type is a reference type to **null**.

```
StringBuffer sb = new StringBuffer("hello");
System.out.println(sb);
// The StringBuffer object is not eligible for collection
sb = null;
// Now the StringBuffer object is eligible for collection
```



Example:

```
class Box {
  double dblWidth;
  double dblHeight;
  double dblDepth;
  Box(){
   //1. default no-argument constructor
       dblWidth=dblHeight=dblDepth=0;
  }
  Box(dbl dblValue){
   // 2. constructor with 1 arg
       dblWidth = dblValue;
       dblHeight = dblValue;
       dblDepth = dblValue;
  }
  Box(double dblWdt, dbl dblHt, dbl dblDpt){
   // 3. constructor with // 3 args
       dblWidth = dblWdt;
       dblHeight = dblHt;
       dblDepth= dblDpt;
  }
  public static void main(String[] args){
     Box  boxObj1 = new Box(); // calls constructor 1
     Box  boxObj2 = new Box(30); // calls constructor 2
     Box  boxObj3 = new Box(24,35,20);  // calls constructor 3
  }}
```



Note :

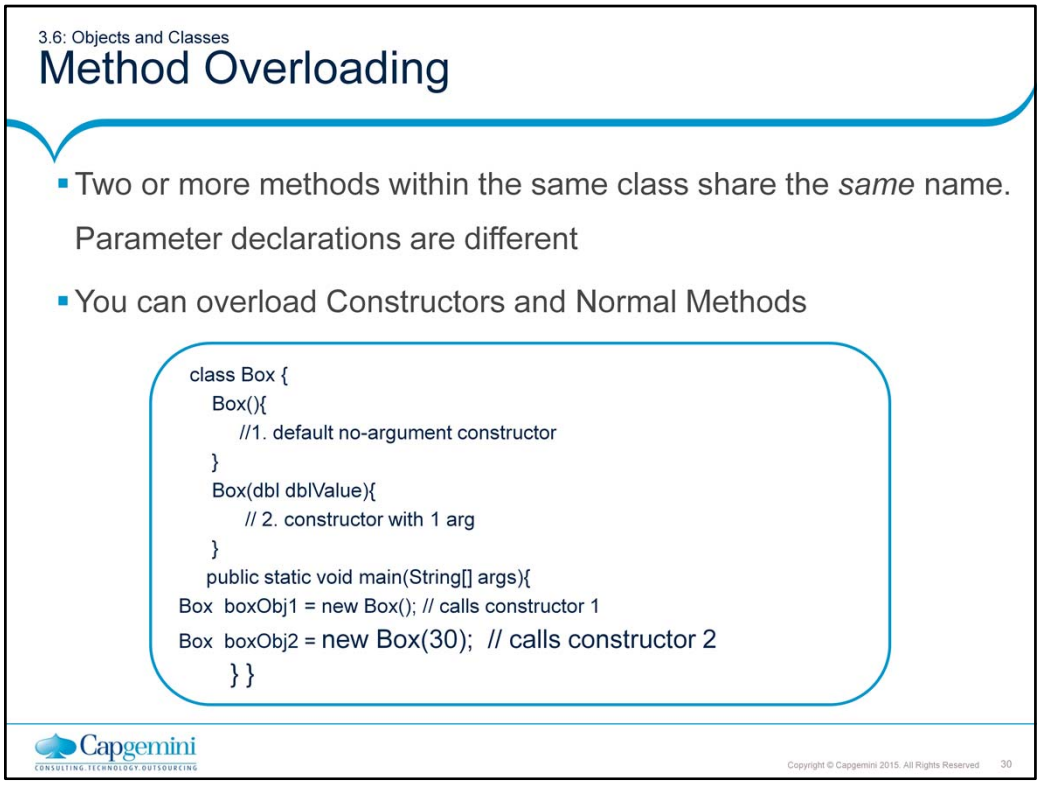
'protected' prevents access to finalize() by code defined outside its class.

This method only approximates the working of C++'s destructor. There is no way to determine when the finalize() method will run. There is no concept of destructors in Java as is there in C++.

To add a *finalizer* to a class, you simply have to override the finalize() method from the object class and can write the code for finalization inside the finalize() method

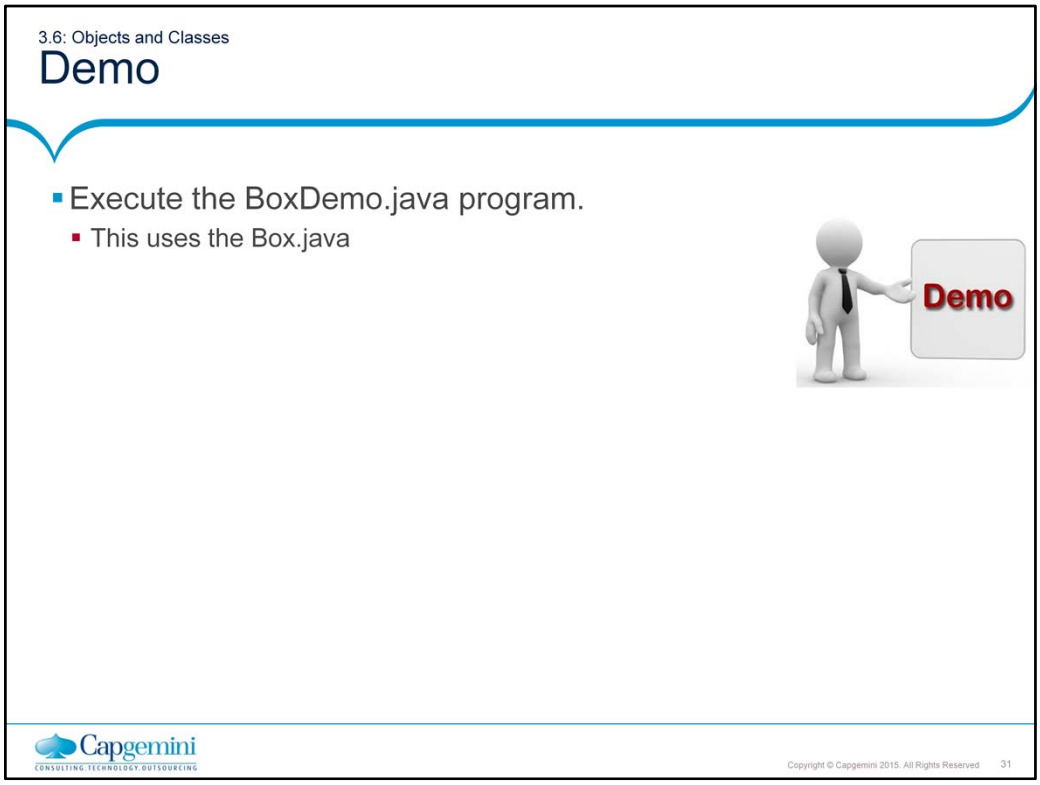
Syntax

```
class A {
  protected void finalize()  {
   super.finalize();
    //Write the code for finalization over here.
    . ...
  }
}
```

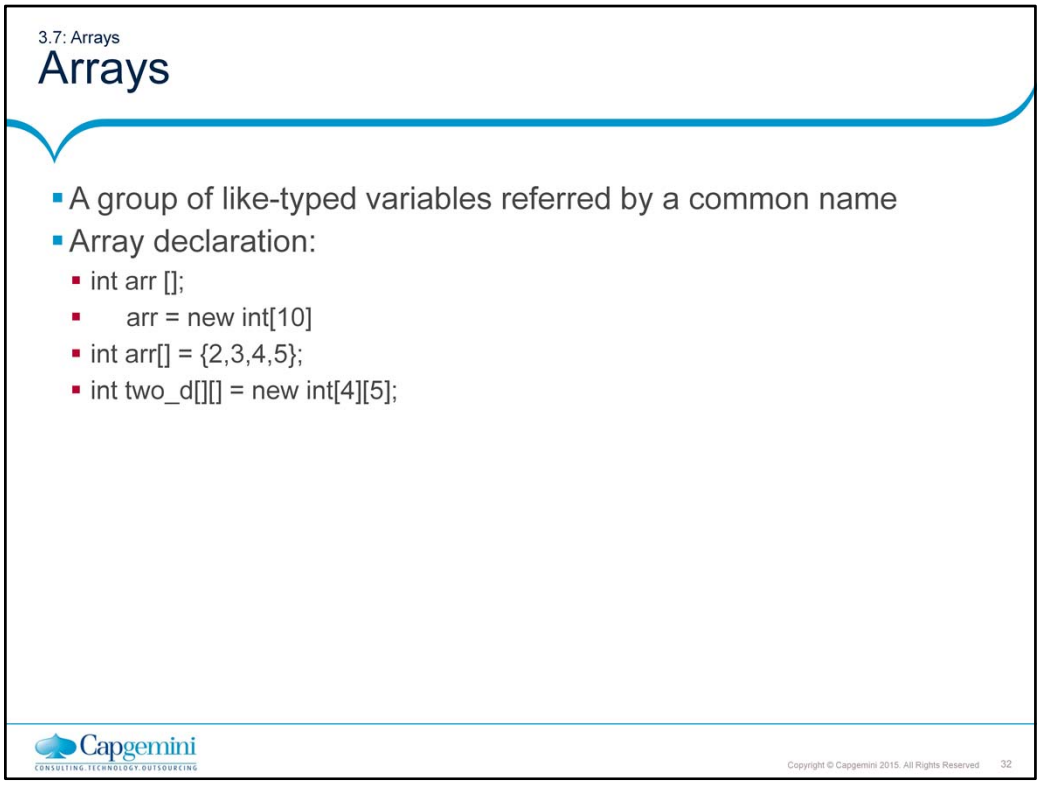


In Java it is possible to define two or more methods within the same class that share the same name, as long as their parameter declarations are different. Here, the methods are said to be overloaded. When an overloaded method is invoked, Java determines which method to actually call by using the type and/or number of operators as its guide.

Refer the example above. Here, two constructors have been declared; one with no arguments, the second with a single argument
Note: In addition to overloading constructors, you can also overload normal methods.

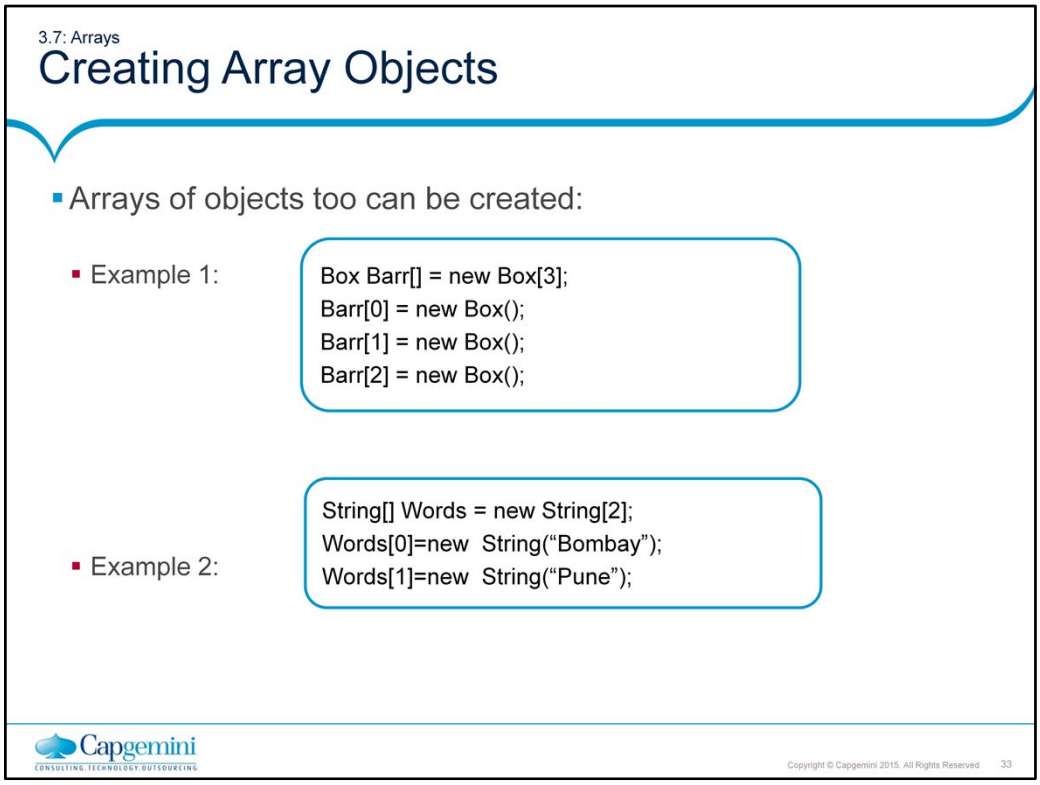


This demo example contains default (no-arg) constructor as well as a constructor that takes 3 parameters. Three box objects are created with initialization done using constructors and setter methods.



Syntax wherein the array is declared and initialized in the same statement:

```
strWords = { "quiet", "success", "joy", "sorrow", "java" };
```



Use new operator or directly initialize an array. When you create an array object using new, all its slots are initialized for you (0 for numeric arrays, false for boolean, '\0' for character arrays, and null for objects).
Like single dimensional arrays, we can form multidimensional arrays as well. Multidimensional arrays are considered as array of arrays in java and hence can have asymmetrical arrays. See an example next.

3.7: Arrays

Demo

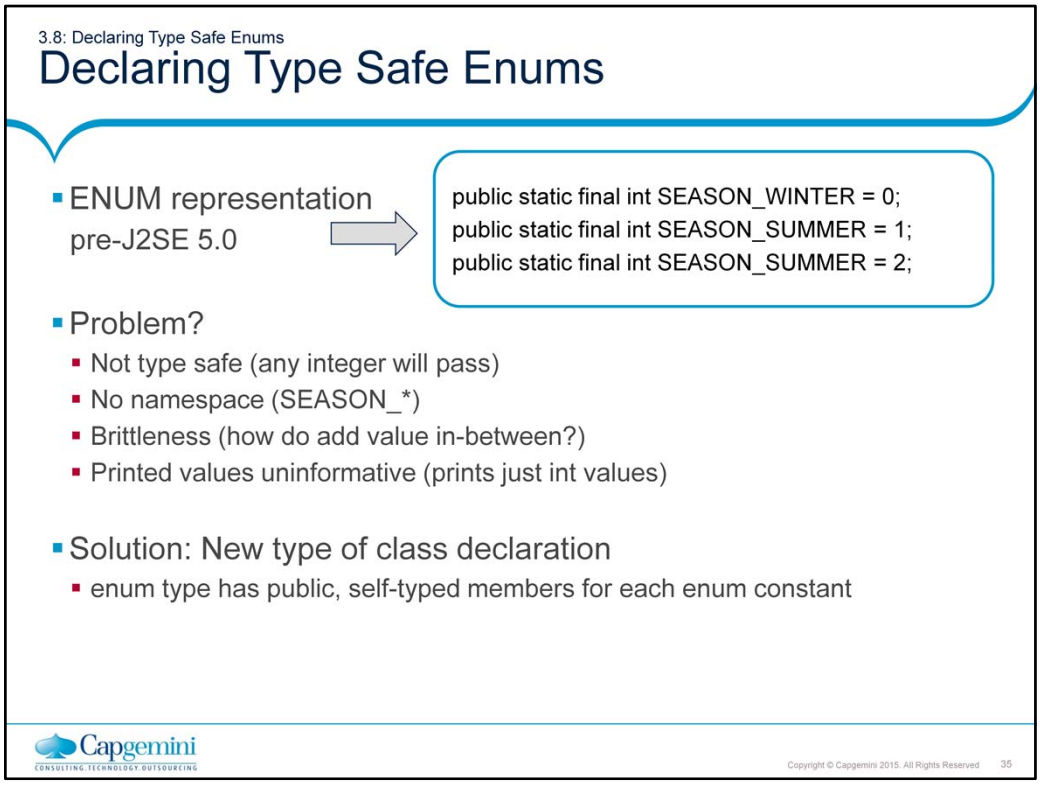
- Executing the ArrayDemo.java program

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 34

We have seen how to pass parameters to program during compile time using parameter passing. One can pass parameters to a program at runtime too. The args parameter (a String array) in main() receives command line arguments.

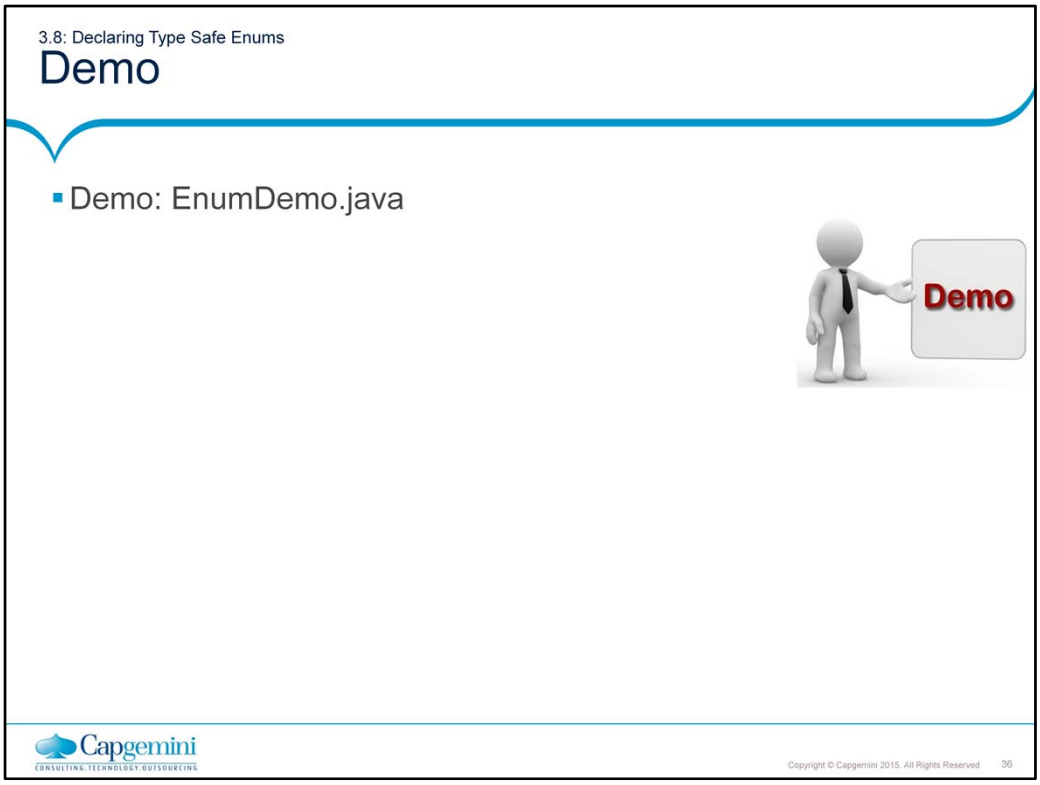
```
class ArrayDemo  {
    int intNumbers[];
    ArrayDemo(int i) {
       intNumbers = new int[i];
    }
    void populateArray() {
         for(int i = 0; i < intNumbers.length; ++i) intnumbers[i] = i;
     }
    void displayContents() {
          for(int i = 0; i <intNumbers.length; ++i)
               System.out.println("Number " + i + ": " + intNumbers[i]);
    }
    public static void main(String[] args)        {
        //Accepting array length as command line argument.
         int intArg = Integer.parseInt(args[0]);
         ArrayDemo ad = new ArrayDemo(intArg);
         ad.displayContents();
         ad.populateArray();
         ad.displayContents();
    } }
```



In pre Java 5, the standard way to represent an enumerated type was the int Enum pattern. An example shown in the box above.

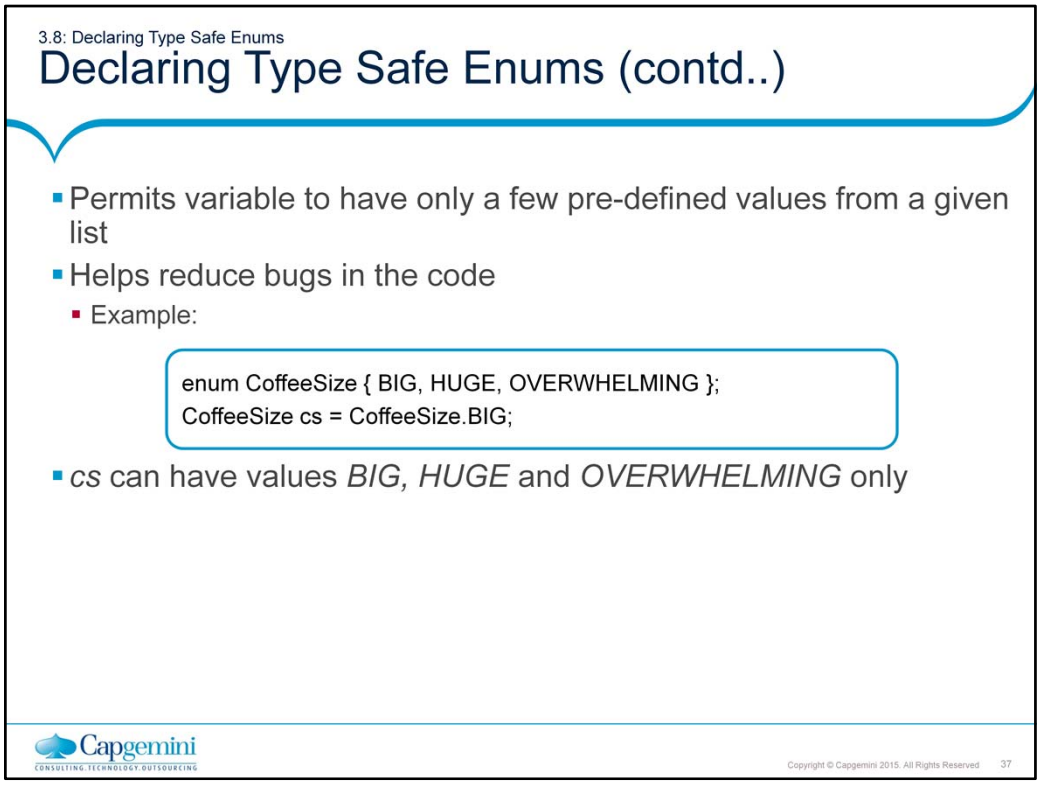
But this pattern suffers from problems such as:
Not typesafe – A season is just an int you can pass in any other int value where a season is required, or add two seasons together (makes no sense)!
No namespace - Constants of an int enum must be prefixed with a string (eg SEASON_) to avoid collisions with other int enum types.
Brittleness - Since int enums are compile-time constants, they are compiled into clients that use them. If a new constant is added between two existing constants or the order is changed, clients must be recompiled.
Printed values are uninformative - Since they are just ints, printing one out will get you a number, which tells you nothing about what it represents, or what type it is.

In J2SE5 the enumerations have become separate classes. Thus, they are type-safe, flexible to use. For example the above enum is now represented as :
enum Season { WINTER, SPRING, SUMMER, FALL }



You can do much more than the plain representation you saw in the previous example. You can define a full-fledged class (dubbed an *enum type*). It not only solves all the problems mentioned previously, it also allows you to add arbitrary methods and fields to an enum type, to implement arbitrary interfaces, and more. Enum types provide high-quality implementations of all the Object methods.

```
enum Apple {
  Jonathan(10, 12), GoldenDel(9, 12), RedDel(12, 5), Winesap(15, 7),
  Cortland( 8, 10);
  private int price,  qty;
   Apple(int p, int x) {   // Constructor
      price = p;
     qty = x;
   }
    int getQty() {  return qty;  }
    int getPrice() { return price; }
}
class EnumDemo {
    public static void main(String args[]) {
    // Display price of Winesap.
        System.out.println("Winesap costs " + Apple.Winesap.getPrice()
                + " cents.\n" + Apple.Winesap.getQty());
  …  … } }
```



Enum lets you restrict a variable to having one of only a few pre-defined values—in other words, one value from an enumerated list. This can help reduce the bugs in your code. For instance, in your coffee shop application you might want to restrict your size selections to BIG, HUGE, and OVERWHELMING. If you let an order for a LARGE or a GRANDE slip in, it might cause an error. See the declaration above. With this, you can guarantee that the compiler will stop you from assigning anything to a CoffeeSize except BIG, HUGE, or OVERWHELMING.
Then, the only way to get a CoffeeSize is with a statement like the following:

```
CoffeeSize cs = CoffeeSize.BIG;
```

It is not required that enum constants be in all upper case, but borrowing from the Sun code convention that constants are named in upper case, it is a good idea.

3.8: Declaring Type Safe Enums

Declare Enums Outside a Class

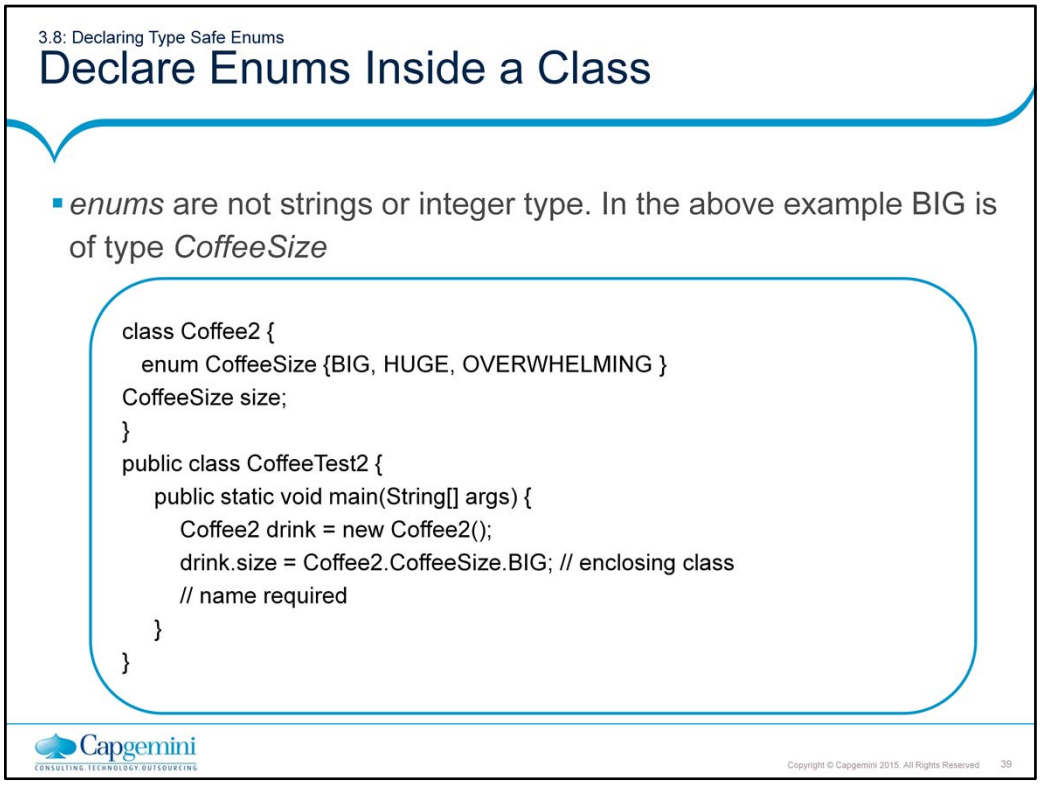
- *enum* can be declared with only a *public* or *default* modifier

```
enum CoffeeSize { BIG, HUGE, OVERWHELMING }
   // this cannot be  private or protected

class Coffee {
      CoffeeSize size;
}
public class CoffeeTest1 {
      public static void main(String[] args) {
            Coffee drink = new Coffee();
             drink.size = CoffeeSize.BIG; // enum outside class
      }
}
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 38



Enums can be declared as their own class, or enclosed in another class, and that the syntax for accessing an enum's members depends on where the enum was declared.

3.8: Declaring Type Safe Enums

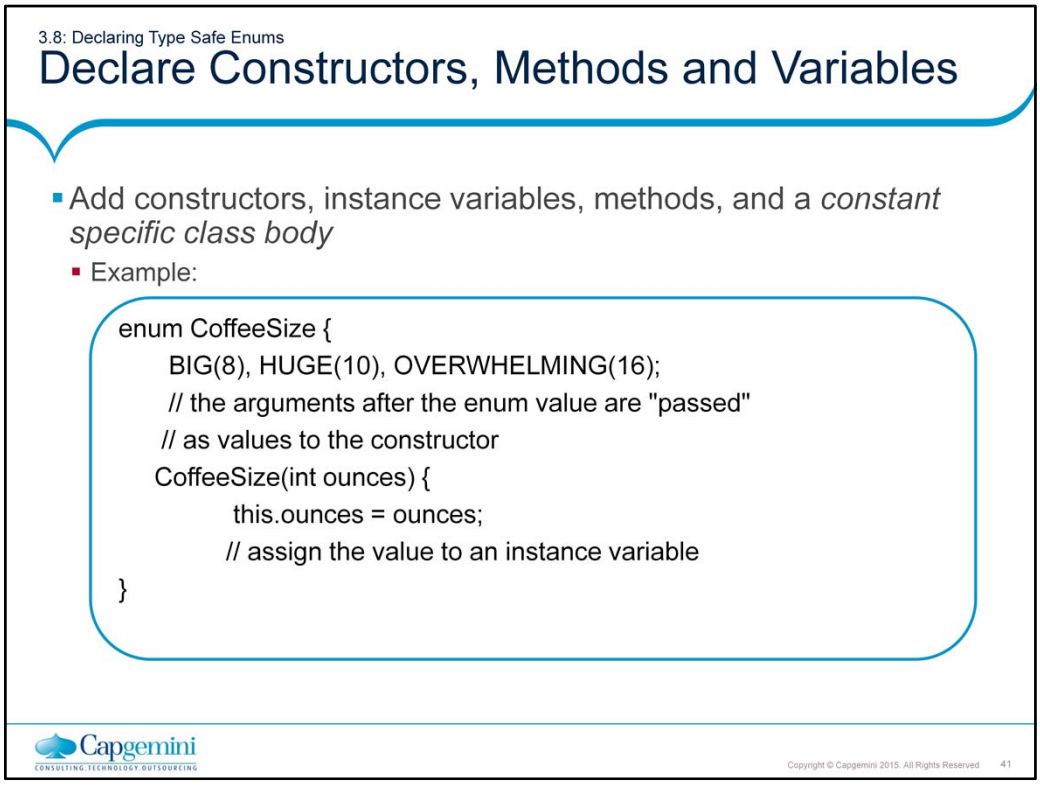
Enums

- *enum* cannot be declared in functions
- A semicolon after an enum is optional

```
public class CoffeeTest1 {
    public static void main(String[] args) {
        enum CoffeeSize { BIG, HUGE, OVERWHELMING }
        // WRONG! Cannot declare enums in methods
        Coffee drink = new Coffee();
        drink.size = CoffeeSize.BIG;
    }
}
```

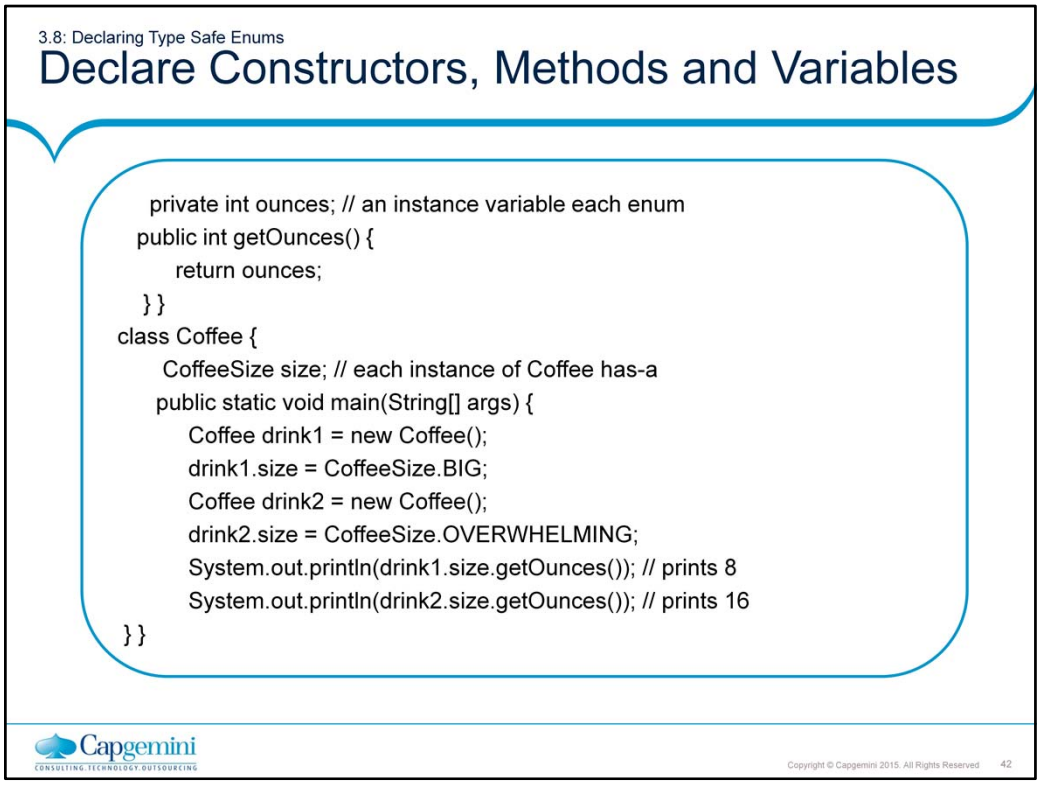
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 40



Because an enum really is a special kind of class, you can do more than just list the enumerated constant values. You can add constructors, instance variables, methods, and something really strange known as a constant specific class body. To understand why you might need more in your enum, think about this scenario: imagine you want to know the actual size, in ounces, that map to each of the three *CoffeeSize* constants.

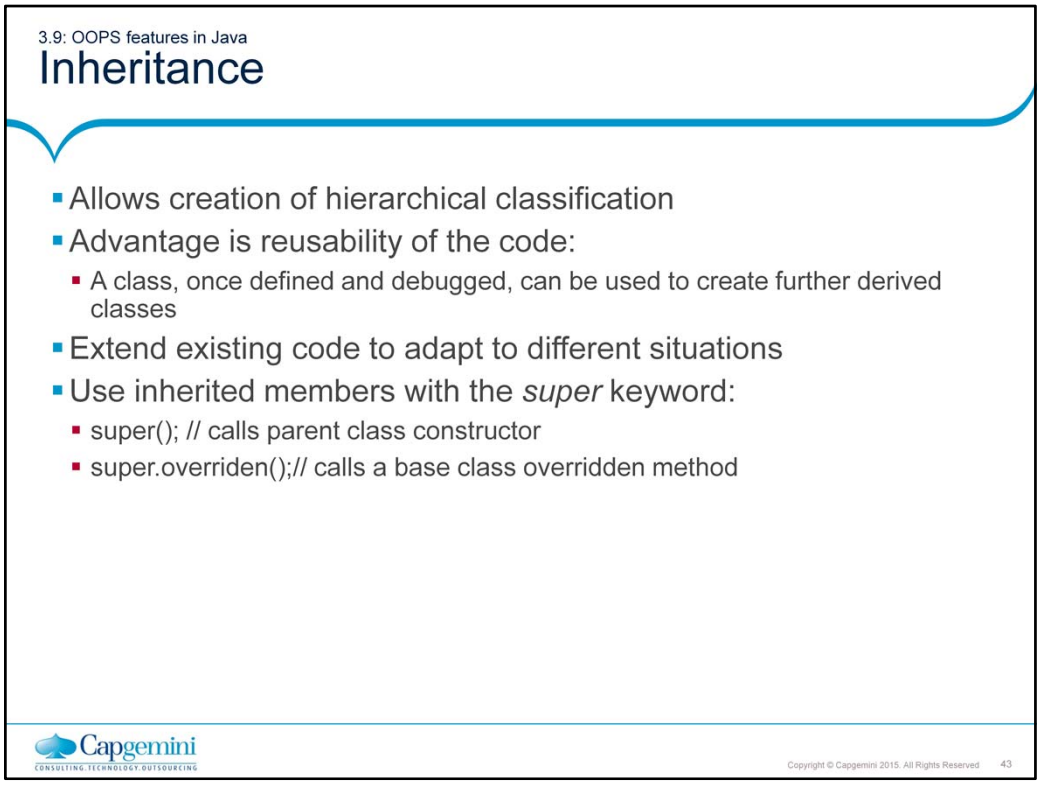
For example, you want to know that BIG is 8 ounces, HUGE is 10 ounces, and OVERWHELMING is a whopping 16 ounces. You could make some kind of a lookup table, using some other data structure, but that would be a poor design and hard to maintain. The simplest way is to treat your enum values (BIG, HUGE, and OVERWHELMING), as objects that can each have their own instance variables. Then you can assign those values at the time the enums are initialized, by passing a value to the enum constructor.



The key points to remember about enum constructors are:

You can NEVER invoke an enum constructor directly. The enum constructor is invoked automatically, with the arguments you define after the constant value. For example, BIG(8) invokes the CoffeeSize constructor that takes an int, passing the int literal 8 to the constructor.

You can define more than one argument to the constructor, and you can overload the enum constructors, just as you can overload a normal class constructor. To initialize a CoffeeType with both the number of ounces and, say, a lid type, you'd pass two arguments to the constructor as BIG(8, "A"), which means you have a constructor in CoffeeSize that takes both an int and a String.



It is one of the fundamental mechanisms for code reuse in object-oriented programming. Inheritance allows new classes to be derived from existing classes. In Java, inheritance specifies how different is the sub class from its parent class. Thus, we can add new variables, methods and also modify the inherited methods. To inherit a class, you simply extend the super class into the subclass. Only single level inheritance is possible in Java.

Super and *super.overridden()* methods are used to invoke base class constructors and base class methods which are overridden in the derived *class.Super()* method allows base class initialization.

3.9: OOPS features in Java

Inheritance

```
class Base {
   public void baseMethod() {
        System.out.println("Base");
   }
}
class Derived extends Base {
    public void derivedMethod() {
       super. baseMethod ();
       System.out.println("Derived");
    }
}
class Test {
    public static void main(String args[]){
            Derived d1=new Derived();
            d1. derivedMethod();
    }
}
```

Output:
Base
Derived

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 44

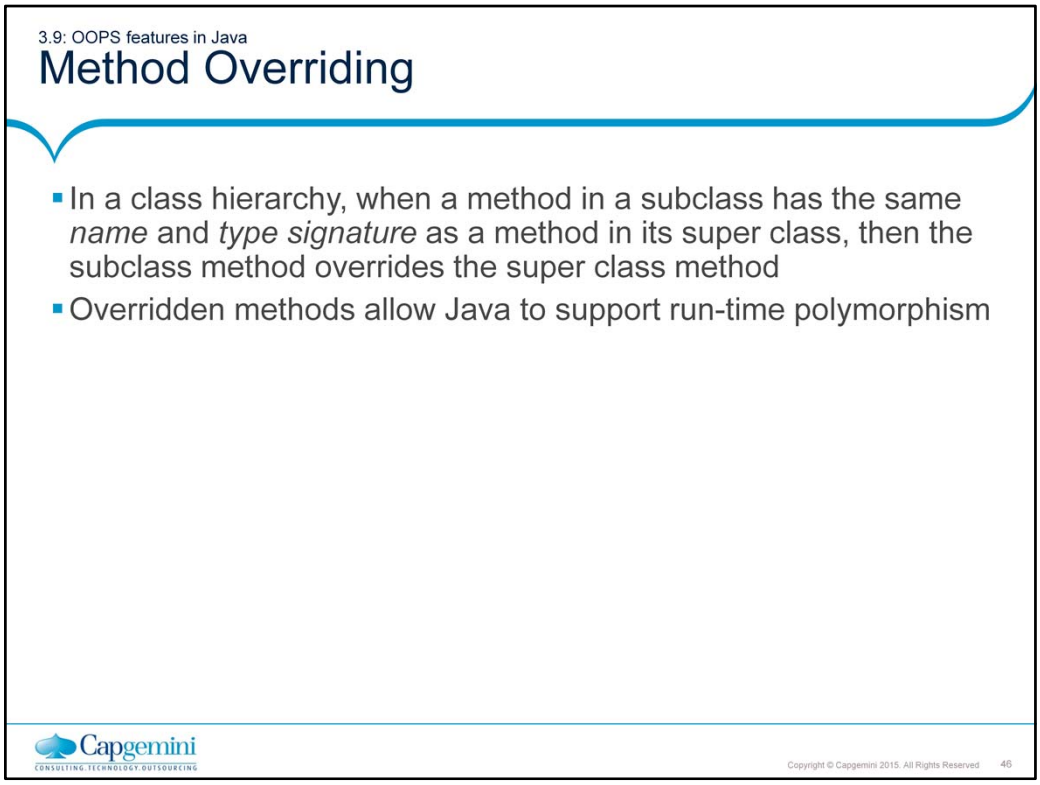
3.9: OOPS features in Java

Polymorphism

- An objects ability to decide what method to apply to itself depending on where it is in the inheritance hierarchy
- Can be applied to any method that is inherited from a *super class*
- Design and implement easily extensible systems

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 45




```
class A{
   void func(){
        // do nothing (dummy) method
    } }
class B extends A{
   void func(){
        System.out.println("In B");
} }
class C extends B{
    void func(){
        System.out.println("In C");
}
public static void main(String args[]){
     A b = new B(); //  creates a new instance of class B at run- time
     b.func();
     A c = new C(); //  creates a new instance of class C at run-time
      c.func();
} }
```

Output:
In B
In C

Lab

- Lab 1
- Lab 2
- Lab 3.1, 3.2, 3.3 and 3.4

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

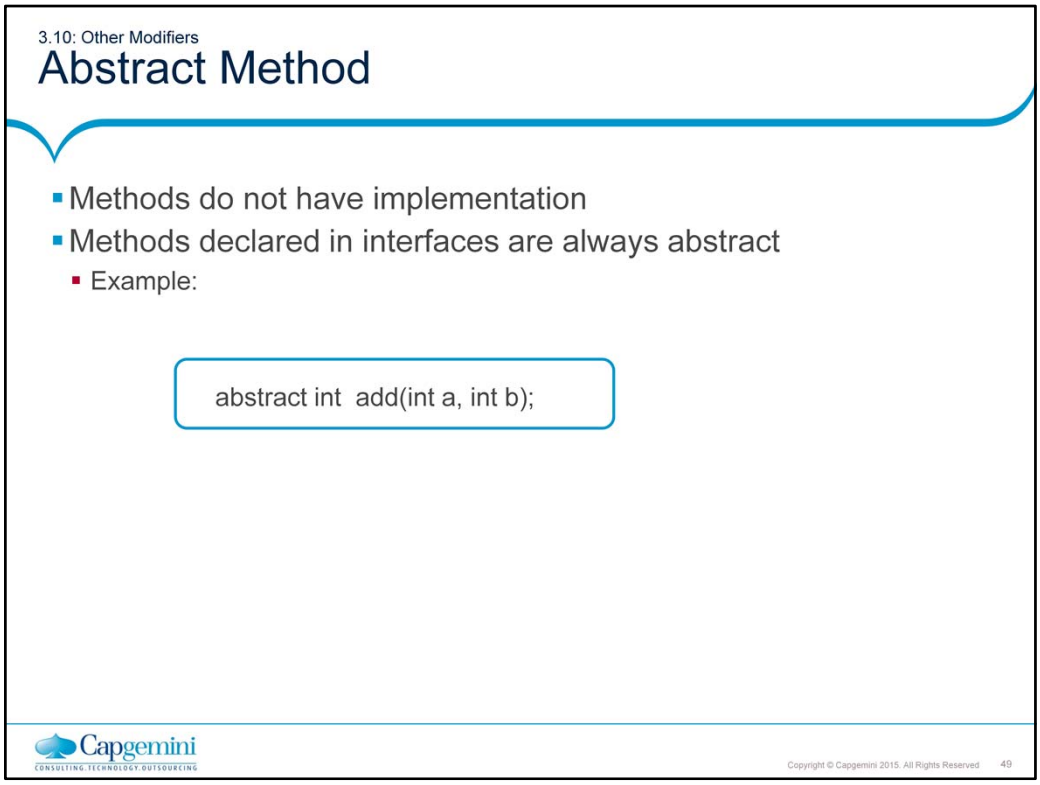
3.10: Other Modifiers

Other Modifiers

- Abstract
  - Method
  - Class
- Final
  - Variable
  - Method
  - Class
- Static
  - Variable
  - method

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 48

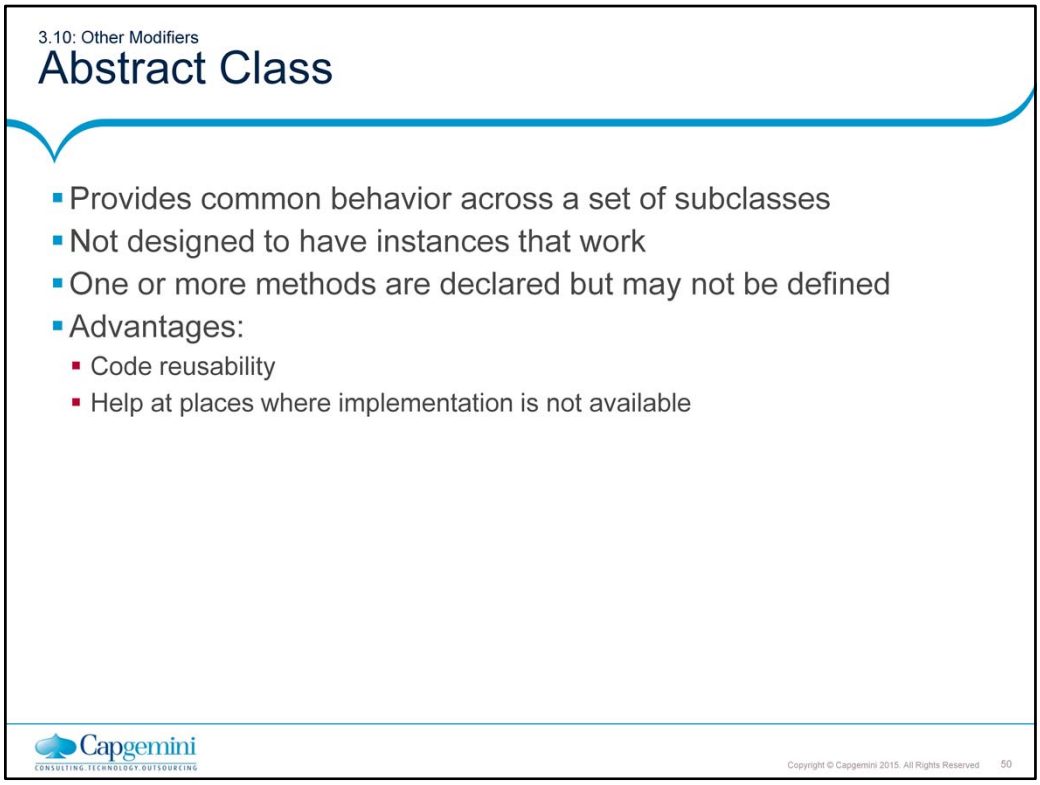


Example of Abstract class – Shape Hierarchy –
You have two hierarchies : Point-Circle-Cylinder and Line-Square-Cube
Here common method is area() – if I create an array that contains the objects of all these classes. How can I do it generically ?
Can Point p = new Line() be valid ? of course not, no inheritance !!!!
So, I force a super class Shape which would contain the method area(). Now what implementation I am going to write in that method. I do not know, at runtime whose area() is going to be called. So, the information that I do not know I put it as “abstract”.

Important points about abstract class :

- You cannot create object of abstract class : why ?
  For example, if a surgeon know how to perform an operation but he does not know how to stitch back the cut stomach will I allow him to touch me?
- An abstract class can contain concrete methods.
- An abstract class may not contain any abstract methods.
- In Java a pure virtual method is called as abstract method.

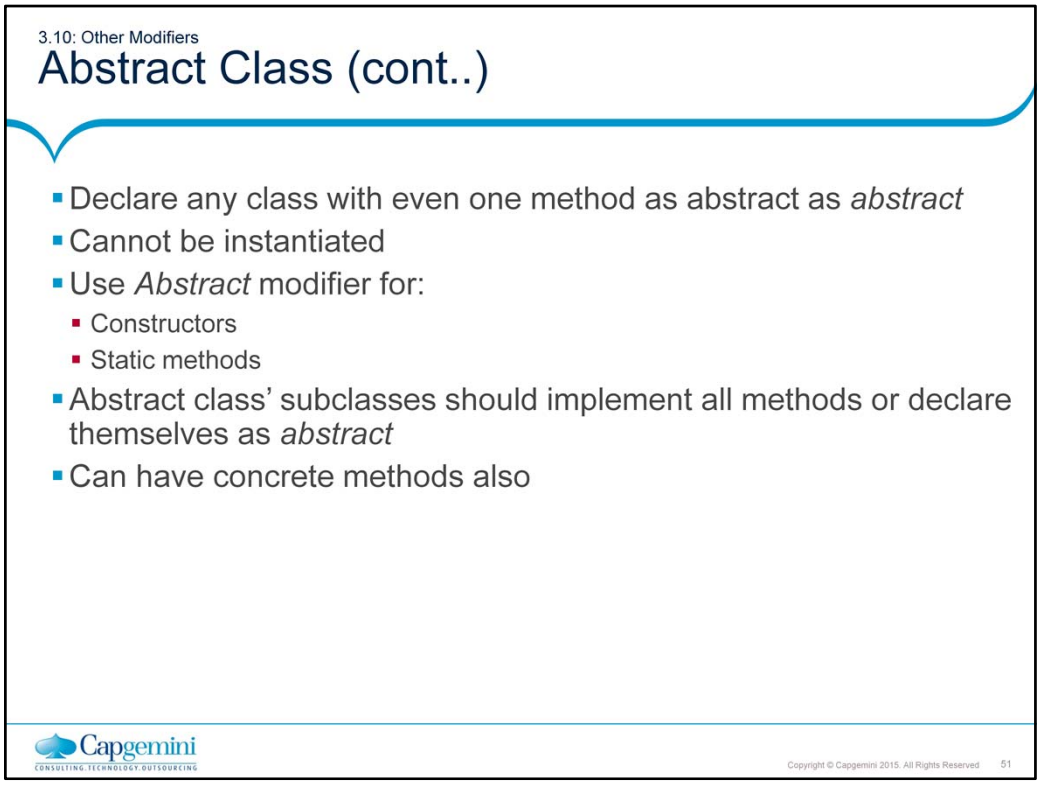
```
abstract class Shape{
public abstract float area();
}
```



In C++, an abstract class is called a pure virtual function & is tagged with a trailing 0.

Example:

```
Class MyClass
{
public:: virtual void area() = 0;
        …..
}
```



Example of Abstract modifier:

```
abstract class Shape{
    abstract void draw();        // observe : no implementation
}

class Rect extends Shape{
     void draw(){      // draw() implemented in subclass Rect
           System.out.println("Drawing a rectangle");
      }
     public static void main(String args[]){
          Shape r1 = new Rect();
          r1.draw();
     }
}
```

Output :
Drawing a rectangle

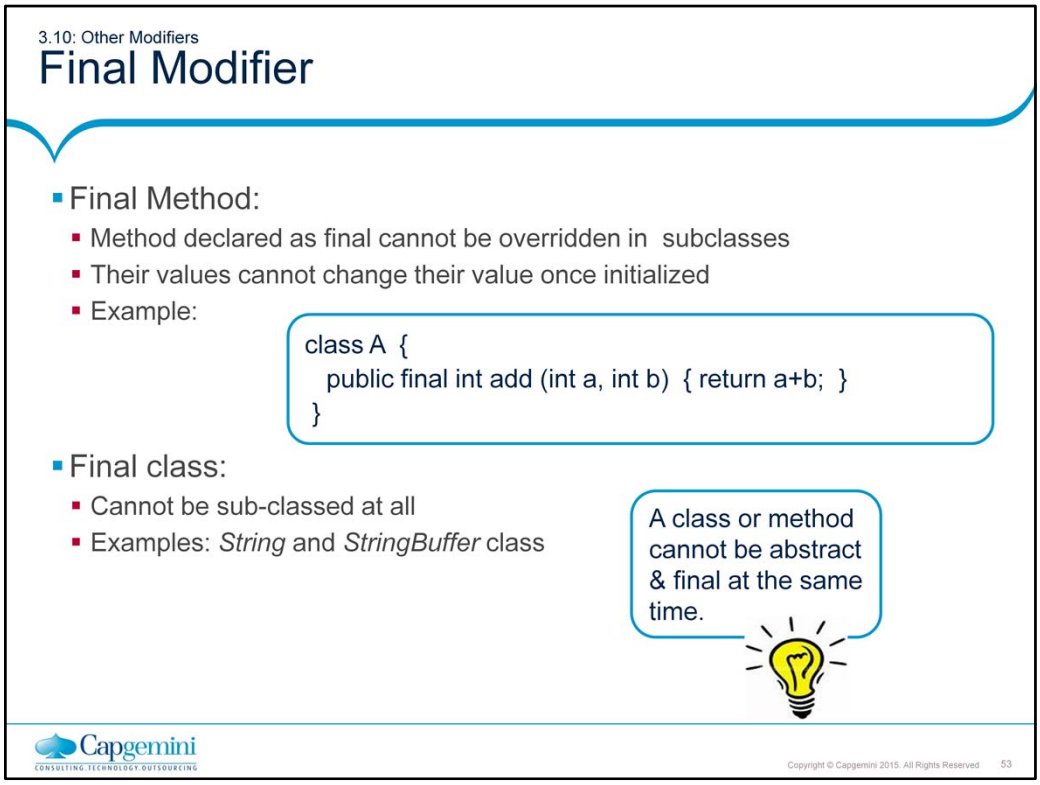
3.10: Other Modifiers

Final Modifier

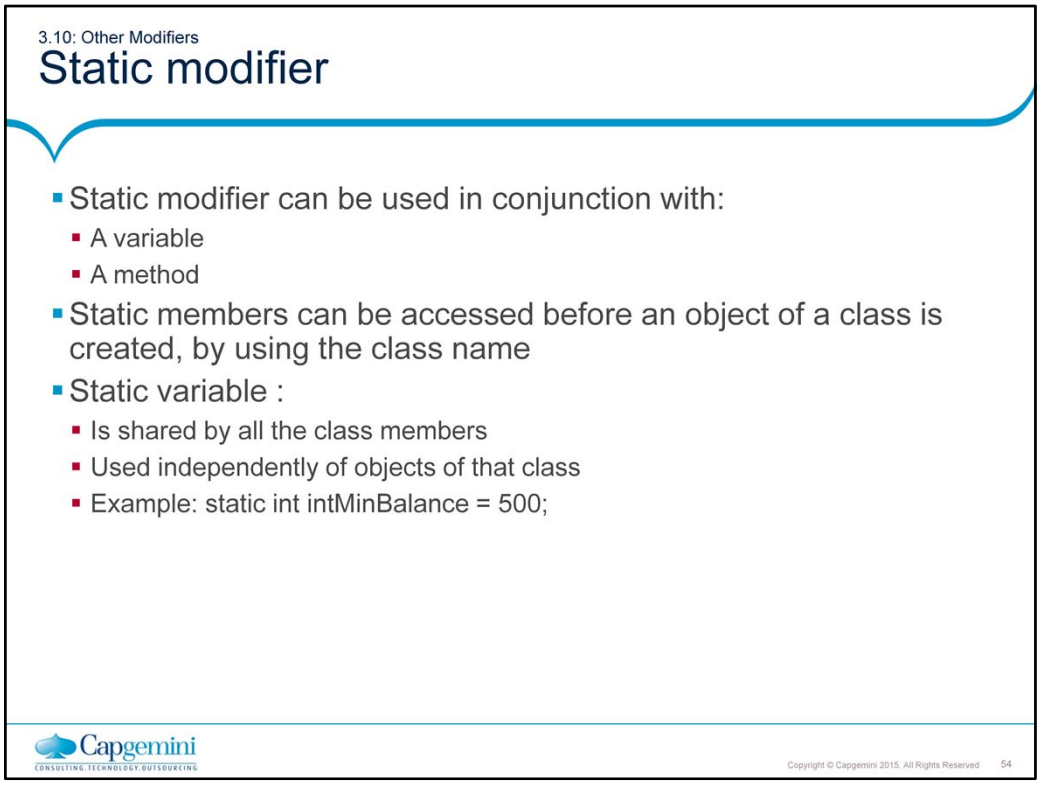
- Final Modifier : Can be applied to variables, methods and classes
- Final variable:
  - Behaves like a constant; ie once initialized, it's value cannot be changed
  - Example: final int i = 10;

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

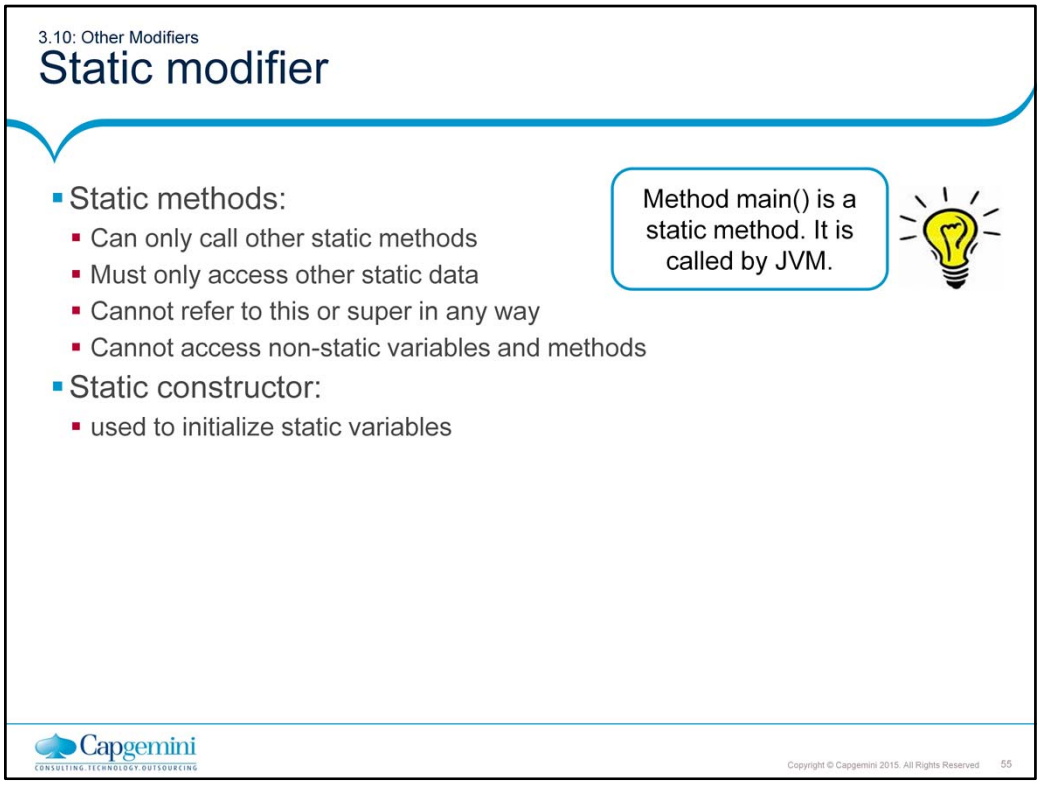
 52



The sub-classes cannot specify a different implementation for final methods. So, final methods cannot be overridden.



Variables and methods marked with static modifier belong to the class rather than any particular instance. Ie, you do not have to instantiate the class to invoke a static method or access a static variable.



main() is called by JVM. Making this method static means the JVM does not have to create an instance of your class to start running code.

Static constructor is also known as static initialization block. This is a normal block of code enclosed in braces { } and preceded by the static keyword. This can appear anywhere in the class body. It is normally used to initialize static variables.

3.10: Other Modifiers

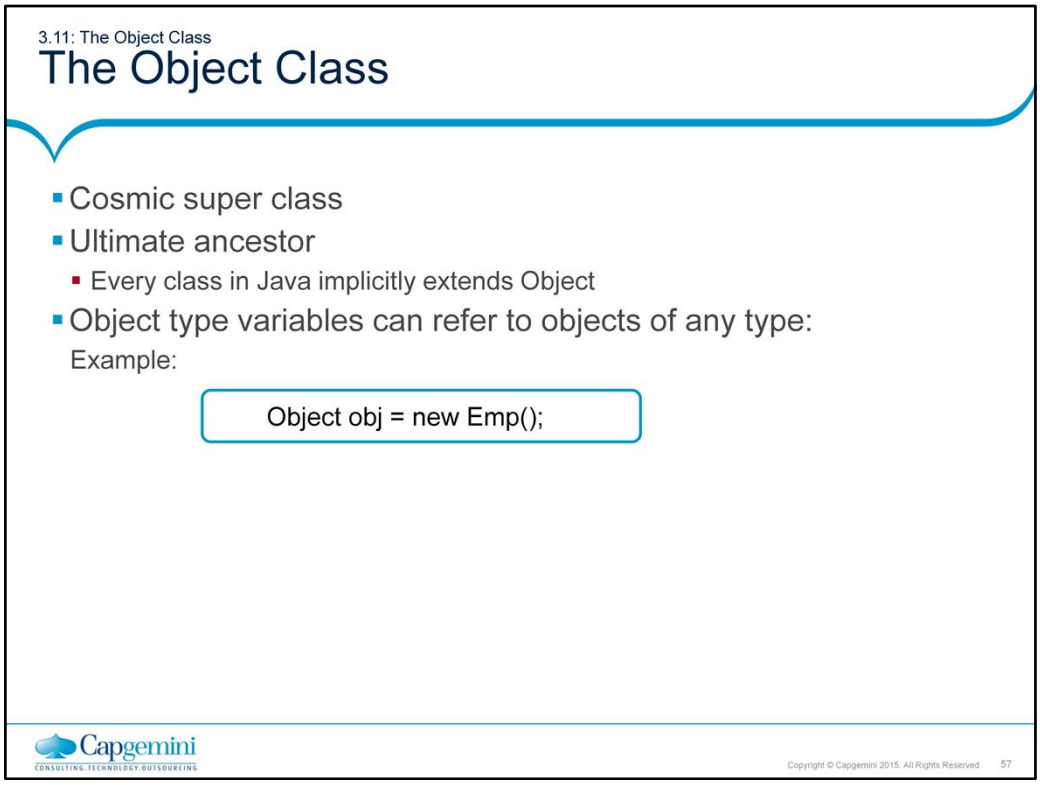
Static modifier

```
// Demonstrate static variables, methods, and blocks.
class UseStatic {
   static int intNum1 = 3;                          // static variable
   static int intNum2;
   static void myMethod(int intNum3) {              // static method
         System.out.println("Number3 = " + intNum3);
         System.out.println("Number1 = " + intNum1);
         System.out.println("Number2 = " + intNum2);
    }
    static {                                        //static constructor
        System.out.println("Static block initialized.");
        intNum2 = intNum1 * 4;
     }
   public static void main(String args[]) {
    myMethod(42);
   }}
```

Output:
Static block initialized.
Number3 = 42
Number1 = 3
Number2 = 12

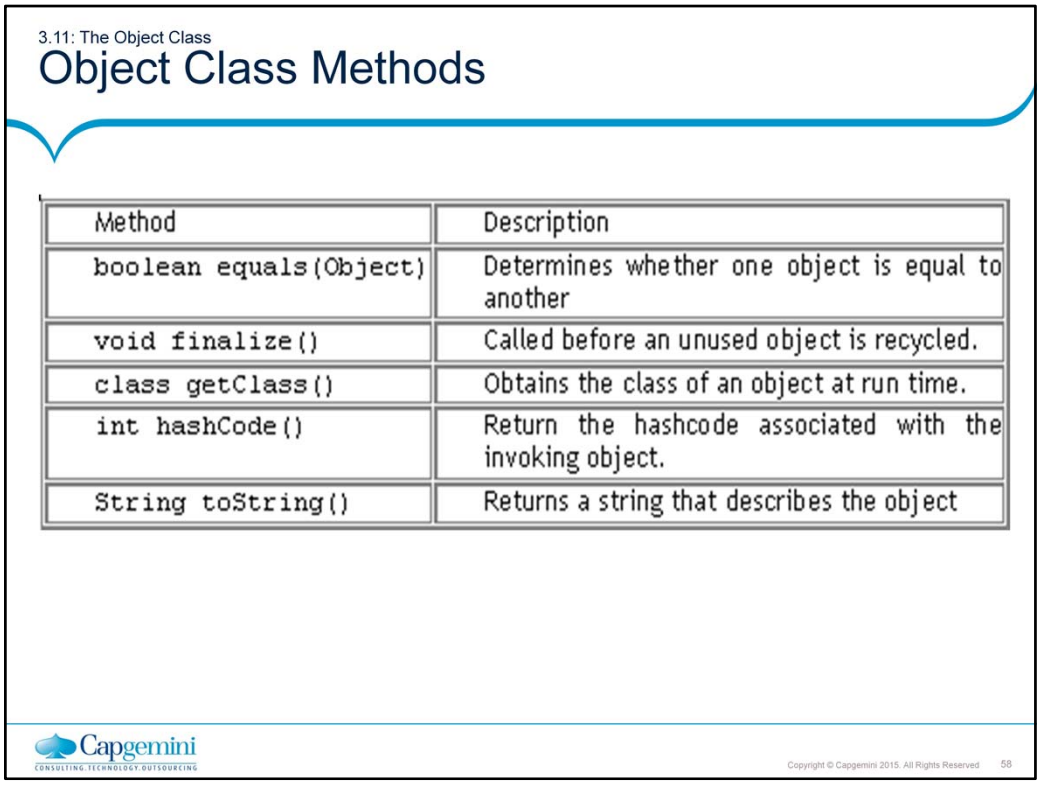
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 56

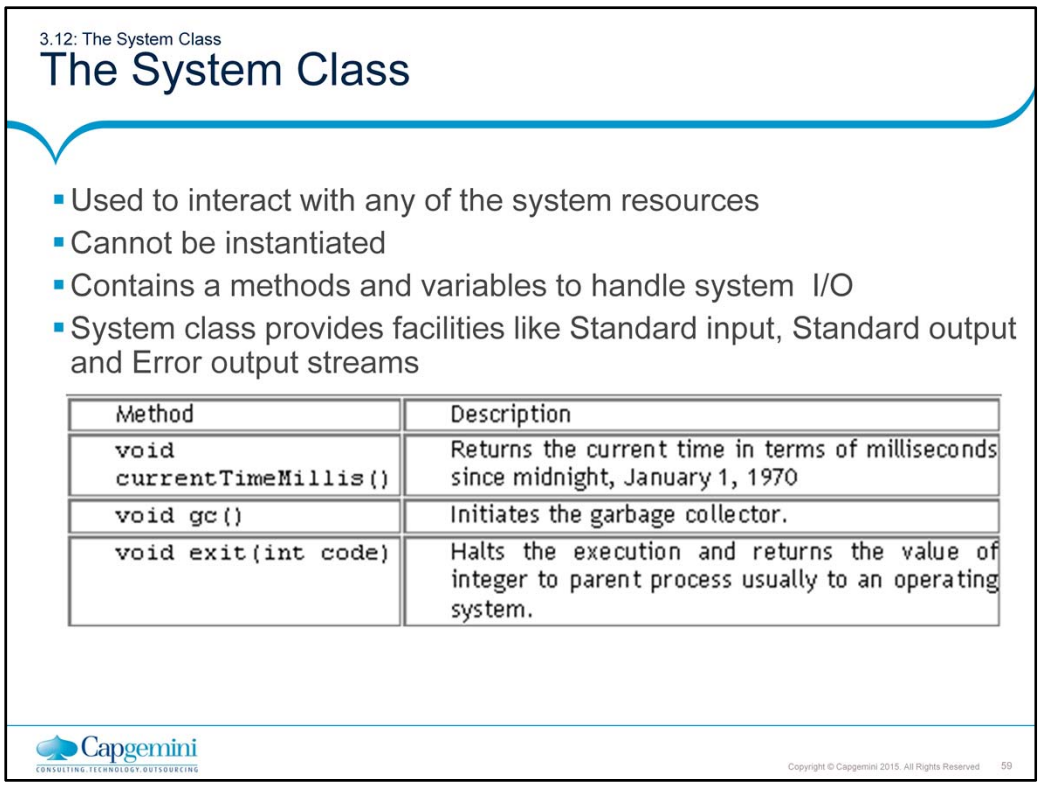


The Object super class is a cosmic super class. Every class in Java extends Object class. It means a reference of object type can refer to an object of any other class. In addition, an array can be referenced by an object reference.
Example:

```
Object Obj1 = new Box(…….);
```



The table above shows some of the methods of the Object superclass. Please refer to Java docs for the entire list.



Note : *System.gc()* is a suggestion and not a command. It is not guaranteed to cause the garbage collector to collect everything.

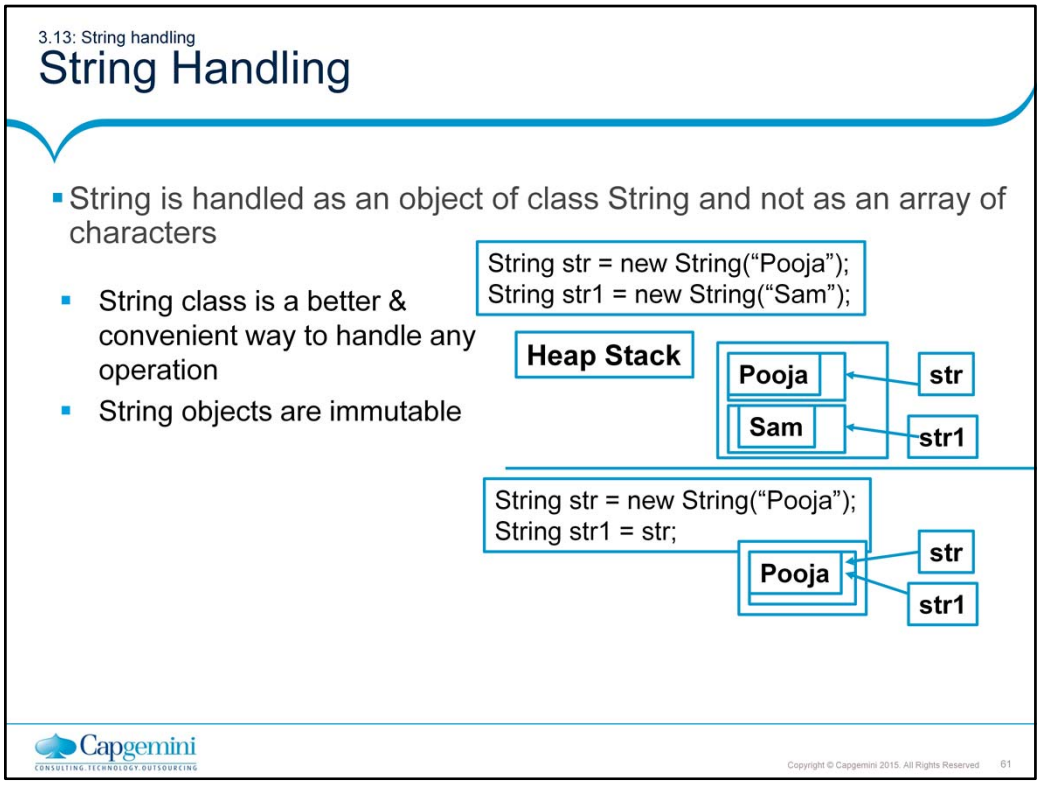
3.12: The System Class

Demo

Execute the Elapsed.java program

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING


```
class Elapsed {
   public static void main(String args[]) throws IOException {
      long start, end;
      int i = 0, sum = 0;
      String str = null;
       System.out.println("Timing a for loop from 0 to 1,000,000");
       // time a for loop from 0 to 1,000,000
      start = System.currentTimeMillis(); // get starting time
      for(int j=0; j < 1000000; j++) ;
     end = System.currentTimeMillis(); // get ending time
     System.out.println("Elapsed time: " + (end-start));
     // Demo to read from the system input and write to standard output.
     BufferedReader br = new
                         BufferedReader(new InputStreamReader(System.in));
      do {
            System.out.println("Enter 0 to quit");
            str = br.readLine();
             i = Integer.parseInt(str);
             if ( i == 0 )   System.exit(0); // normal exit
             sum += i;
             System.out.println("Current sum is: " + sum);
       } while(i != 0);
}}
```



String is not an array of characters but it is actually a class and is part of core API. We can use the class String as a usual data-type. It can store up to 2 billion characters.
Note: A String in java is not equivalent to character array.
Strings are built-in objects & thus have a full complement of features that make string handling convenient. For example, Java has methods to compare two strings, search for a sub string, concatenate two strings etc. In addition, String objects can be constructed in number of ways.

String objects are immutable objects. That is, once you create a String object you cannot change the characters, which are part of String. This seems to be a major restriction, but that is not the case. Every time you perform some modification operation on the object, a new String object is created that contains the modifications. The original string is left unchanged. Hence, the number of operations performed on one particular string creates those many string objects.

For those cases in which modifiable string is desired, there is a companion class to String called **StringBuffer**, whose objects contain strings that can be modified after they are created.

3.13: String Handling

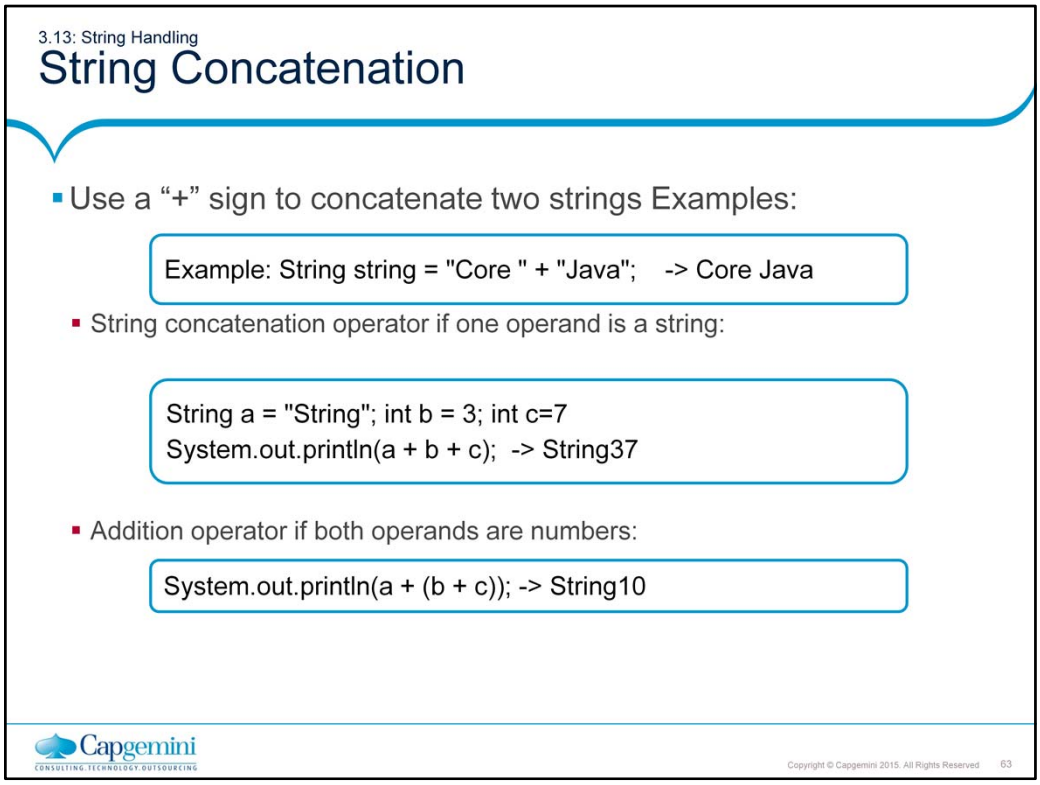
Important Methods

- length(): length of string
- indexOf(): searches an occurrence of a char, or string within other string
- substring(): Retrieves substring from the object
- trim(): Removes spaces
- valueOf(): Converts data to string
- concat(String s) : Used to concatenate a string to an existing string. Eg

```
String string = "Core ";
System.out.println( string=string.concat(" Java") );
Output -> "Core Java"
```

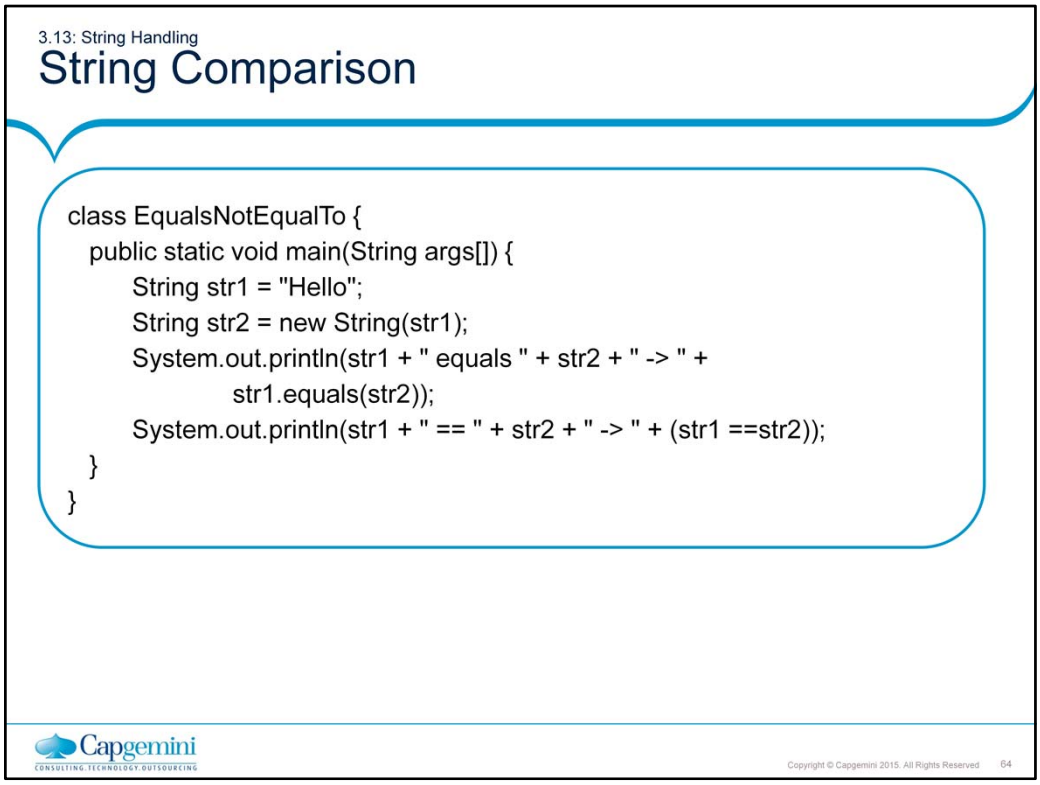
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 62



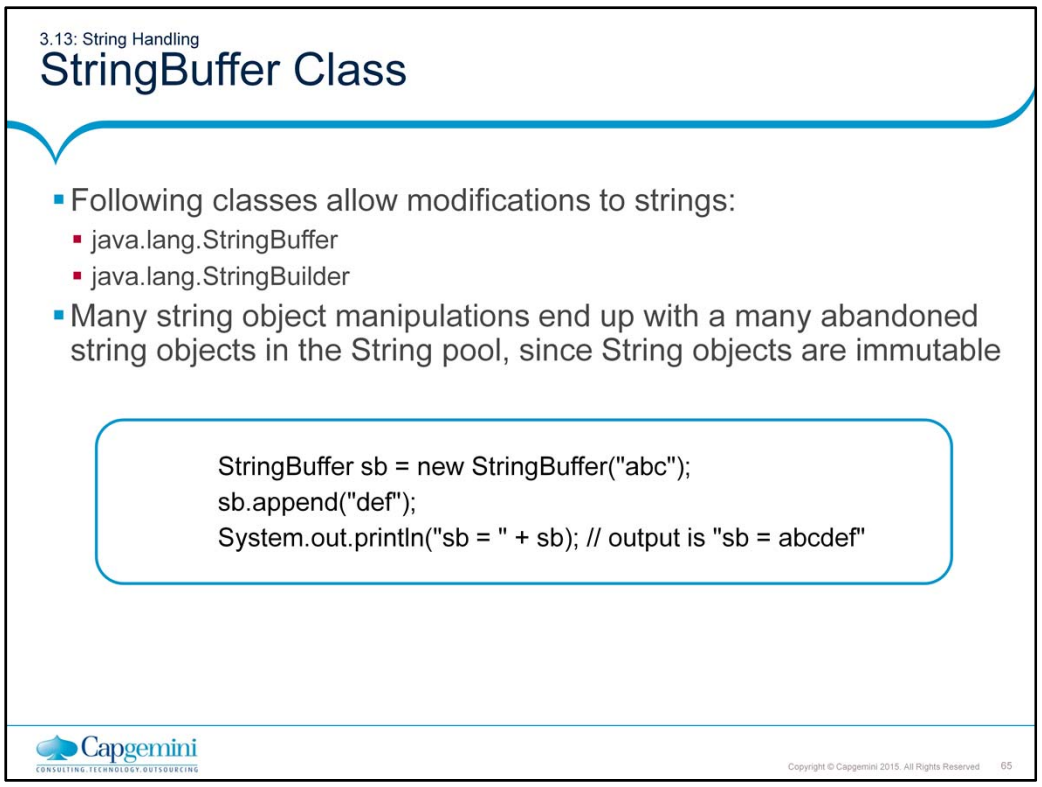
The concat() method seen in previous page allows one string to be concatenated to another. But Java also supports string concatenation with the “+” operator.
In general, Java does not support operator overloading. The exception to this rule is the + operator, which concatenates two strings, and produces a new string object as a result.

```
class SimpleString  {
    public static void main(String args[])  {
        // Simple String Operations
        char c[] = {'J', 'a', 'v', 'a'};
        String s1 = new String(c); // String constructor using
        String s2 = new String(s1);
        // String constructor using string as arg.
        System.out.println(s1);
        System.out.println(s2);
        System.out.println("Length of String s2 : " + s2.length());
        System.out.println("Index of v : " + s2.indexOf('v'));
        System.out.println("s2 in uppercase : " + s2.toUpperCase());
        System.out.println("Character at position 2 is  : " +  s2.charAt(1));
        // Using concatenation to prevent long lines.
         String longStr = "This could have been " +
                          "a very long line that would have " +
                          "wrapped around.  But string concatenation " +
                          "prevents this.";
         System.out.println(longStr);
    } }
```



The String class includes various methods that compare strings or substrings within each string. The most popularly used two ways to compare the strings is either using = = operator or by using the equals method.

The equals() method compares the characters inside a String object. The = = operator compare two object references to see whether they refer to the same instance. The program above shows the difference between the two.



Let us understand StringBuilder with an example.

```
String x = "abc";
x.concat("def");
System.out.println("x = " + x); // output is "x = abc"
```

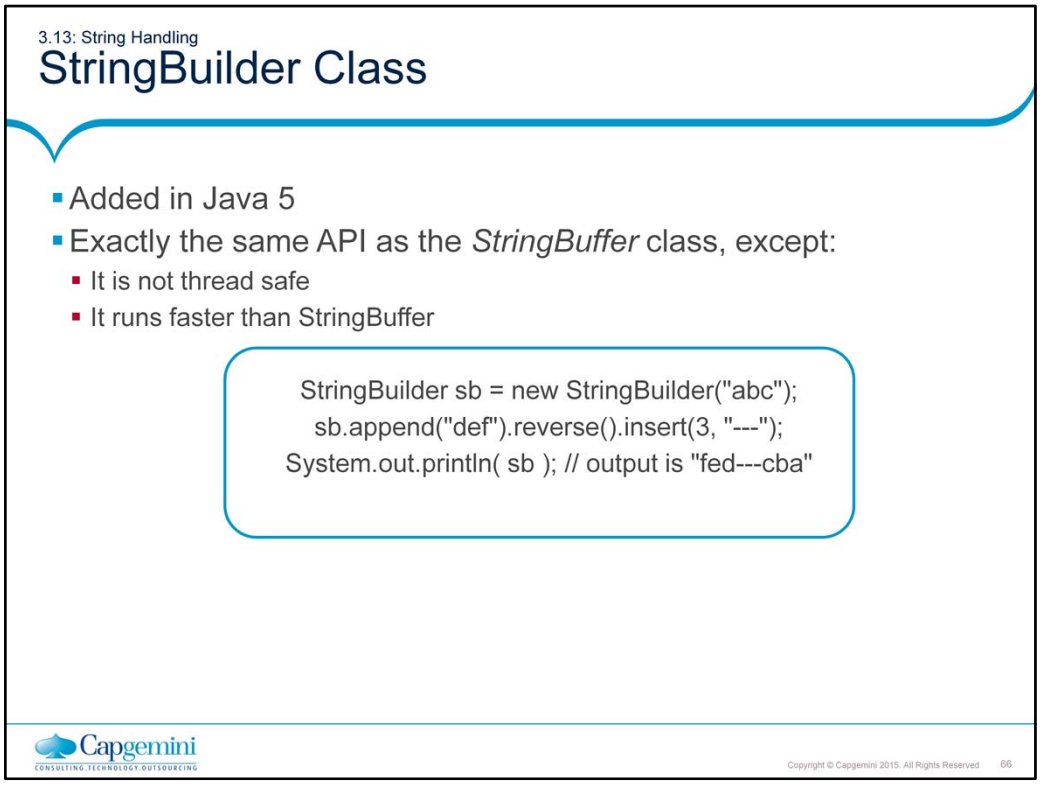
Because no new assignment was made, the new String object created with the concat() method was abandoned instantly. We also saw examples like this:

```
String x = "abc";
x = x.concat("def");
System.out.println("x = " + x); // output is "x = abcdef"
```

We got a nice new String out of the deal, but the downside is that the old String "abc" has been lost in the String pool, thus wasting memory. If we were using a StringBuffer instead of a String, the code would look like this:

```
StringBuffer sb = new StringBuffer("abc");
        sb.append("def");
        System.out.println("sb = " + sb); // output is "sb = abcdef“
```

Note: Refer Javadocs to know more about other methods of StringBuilder.



The StringBuilder class was added in Java 5. It has exactly the same API as the StringBuffer class, except StringBuilder is not thread safe. In other words, its methods are not synchronized. Sun recommends that you use StringBuilder instead of StringBuffer whenever possible because StringBuilder will run faster (and perhaps jump higher). So, apart from Synchronization, anything we say about StringBuilder's methods holds true for StringBuffer's methods, and vice versa.

Note: Refer Javadocs to know more about methods of StringBuilder.

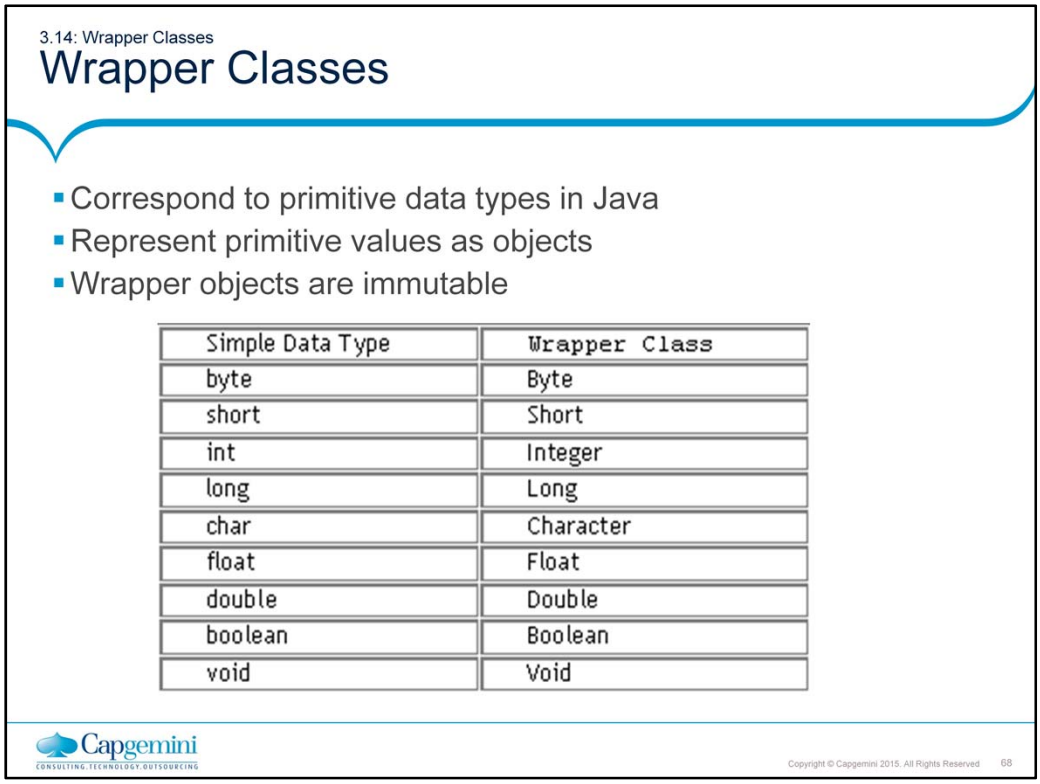
3.13: String Handling

Demo

- Execute the following programs:
  - SimpleString.java
  - ToStringDemo.java
  - StringBufferDemo.java
  - CharDemo.java

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 67



Wrapper classes correspond to the primitive data types in the Java language. These classes represent the primitive values as objects. Wrapper objects are immutable. This means that once a wrapper object has a value assigned to it, that value cannot be changed.
Java uses simple or primitive data types, such as int, char and Boolean etc. These data types are not part of the object hierarchy. They are passed by value to methods and cannot be directly passed by reference. However, at times there is a need to create an object representation of these simple data types. Java provides classes that correspond to each of these simple types.These classes encapsulate, or wrap, the simple data type within a class. Thus, they are commonly referred to as wrapper classes. The abstract class Number defines a superclass that is implemented by all numeric wrapper classes.

```
class IsDemo {
public static void main(String args[]) {
char a[] = {'a', 'b', '5', '?', 'A', ' '};
for(int i=0; i<a.length; i++) {
if(Character.isDigit(a[i]))
System.out.println(a[i] + " is a digit.");
if(Character.isLetter(a[i]))
System.out.println(a[i] + " is a letter.");
if(Character.isUpperCase(a[i]))
System.out.println(a[i] + " is uppercase.");
if(Character.isLowerCase(a[i]))
System.out.println(a[i] + " is lowercase.");
} } }
```

3.14: Wrapper Classes

Casting for Conversion of Data type

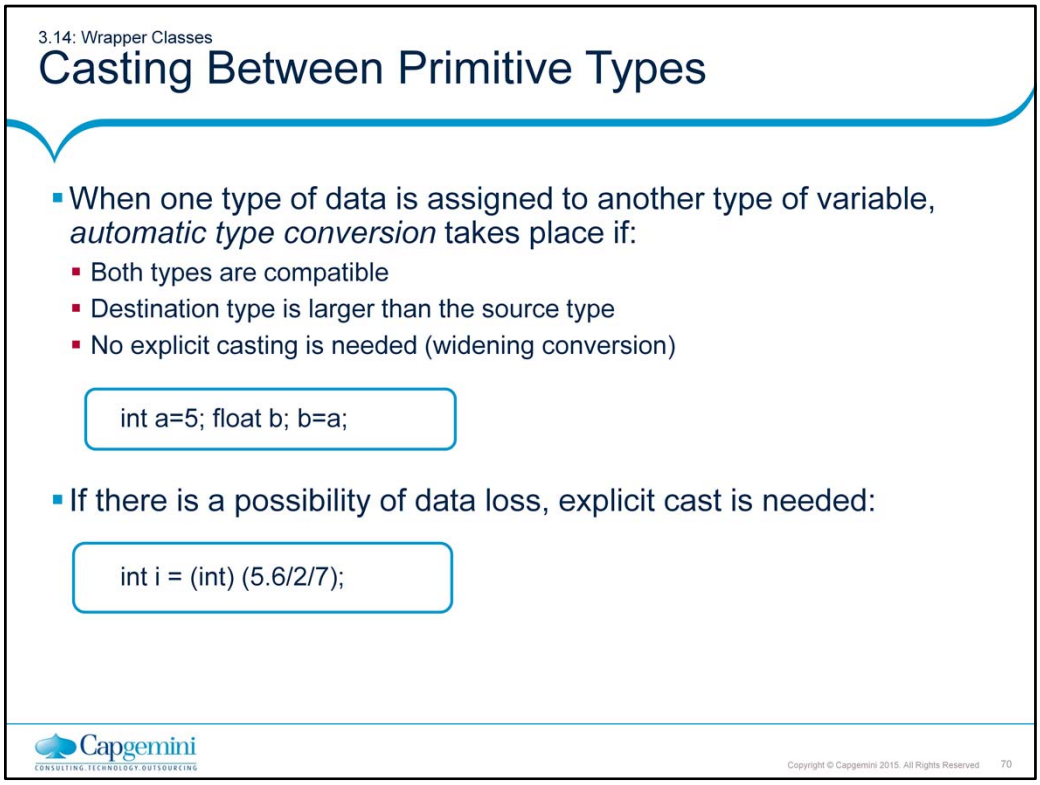
- Casting operator converts one variable value to another where two variables correspond to two different data types

```
variable1 = (variable1) variable2
```

- Here, *variable2* is typecast to *variable1*
- Data type can either be a *reference* type or a *primitive* one

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 69



When one type of data is assigned to another type of variable, an automatic type conversion will take place if the following two conditions are met:

- The two types are compatible.
- The destination type is larger than the source type.

In this case, a widening conversion takes place. For example, the int type is always large enough to hold all valid byte values, so no explicit cast statement is required.

To widen conversions, numeric types, including integer and floating-point types, are compatible with each other. However, numeric types are not compatible with char or boolean. Also, char and boolean are not compatible with each other.

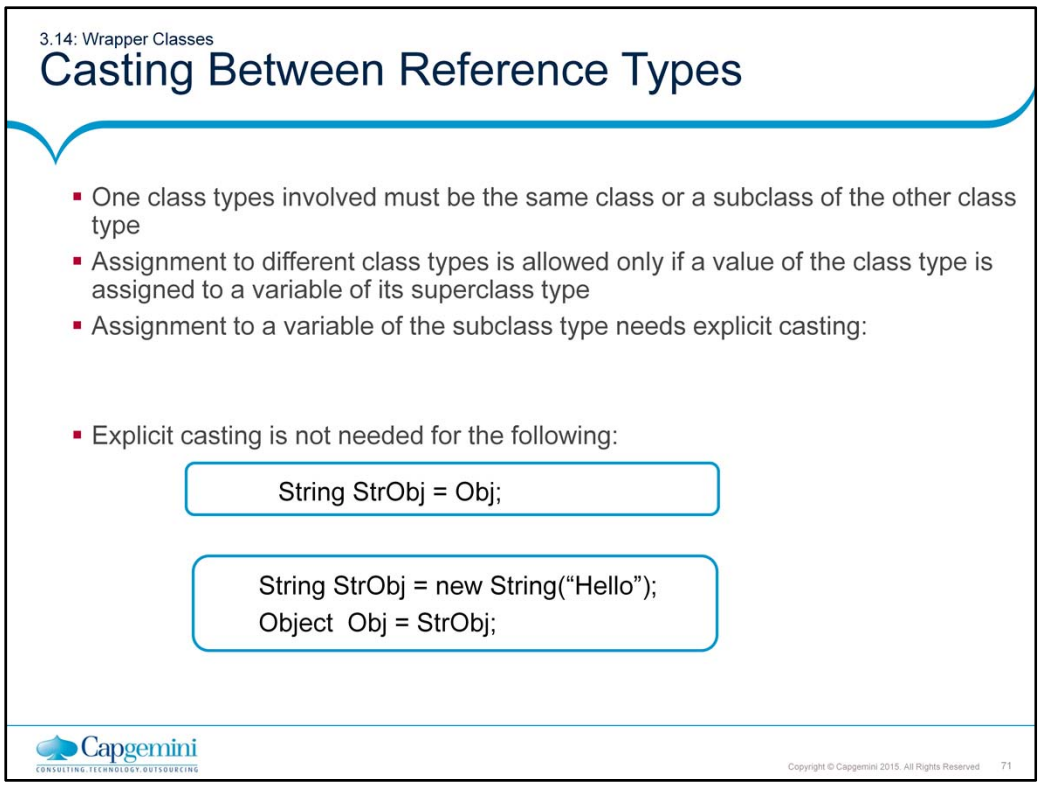
As mentioned earlier, Java also performs an automatic type conversion when storing a literal integer constant into variables of type byte, short, or long.

Casting Incompatible Types

Automatic type conversions may not fulfill all needs though. For example, assigning an int value to a byte variable. This conversion will not be performed automatically, because a byte is smaller than an int. This is called a *narrowing* conversion, since you are explicitly making the value narrower so that it will fit into the target type. This is done using a *cast* - an explicit type conversion. It has this general form: (target-type) value

Here, target-type specifies the desired type to convert the specified value to. For example, the following fragment casts an int to a byte:

```
int i = 125;  byte b;
b = (byte) i;
```



The first rule of casting between reference types is that one of the class types involved must be the same class as, or a subclass of, the other class type. Assignment to different class type is allowed only if a value of the class type is assigned to a variable of its superclass type. Assignment to a variable of the subclass type needs explicit casting.

```
Object Obj = new Object ( );
String StrObj = Obj;
/*    The second statement of the above code is not a legitimate
one, because class String is a subclass of class object.  So, an
explicit casting is needed. This is called as downcasting
*/
String StrObj = (String) Obj;
//The reverse statement will not require casting. It is upcasting
String StrObj = new String("Hello");
Object  Obj = StrObj;
```

Rule number one in casting is that you cannot cast a primitive type to a reference type, nor can you cast the other way around. If a casting violation is detected at runtime, the exception ***ClassCastException*** is thrown.

3.14: Wrapper Classes

Casting Between Reference Types (contd..)

- Two types of reference variable castings:
  - Downcasting:

```
Object Obj = new Object ( );
String StrObj = (String) Obj;
```

  - Upcasting:

```
String StrObj = new String("Hello");
Object  Obj = StrObj;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 72

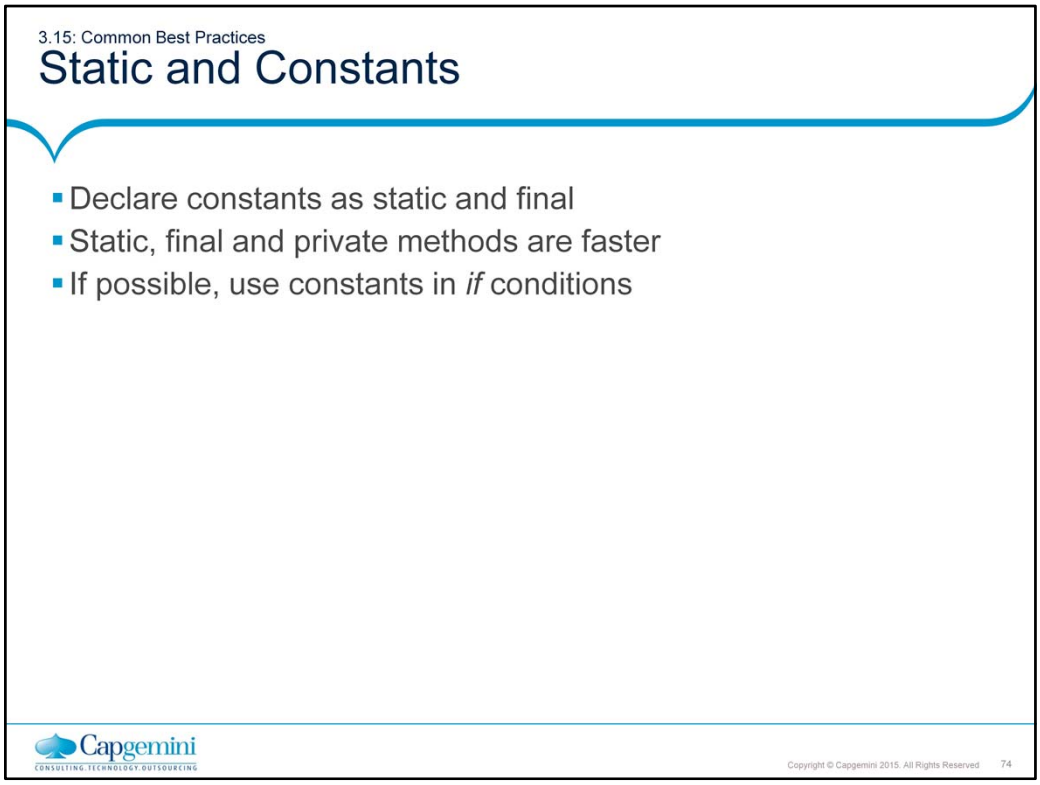
3.14: Simple Formatted I/O and Scanner

Lab

- Lab 3.5, 3.6

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

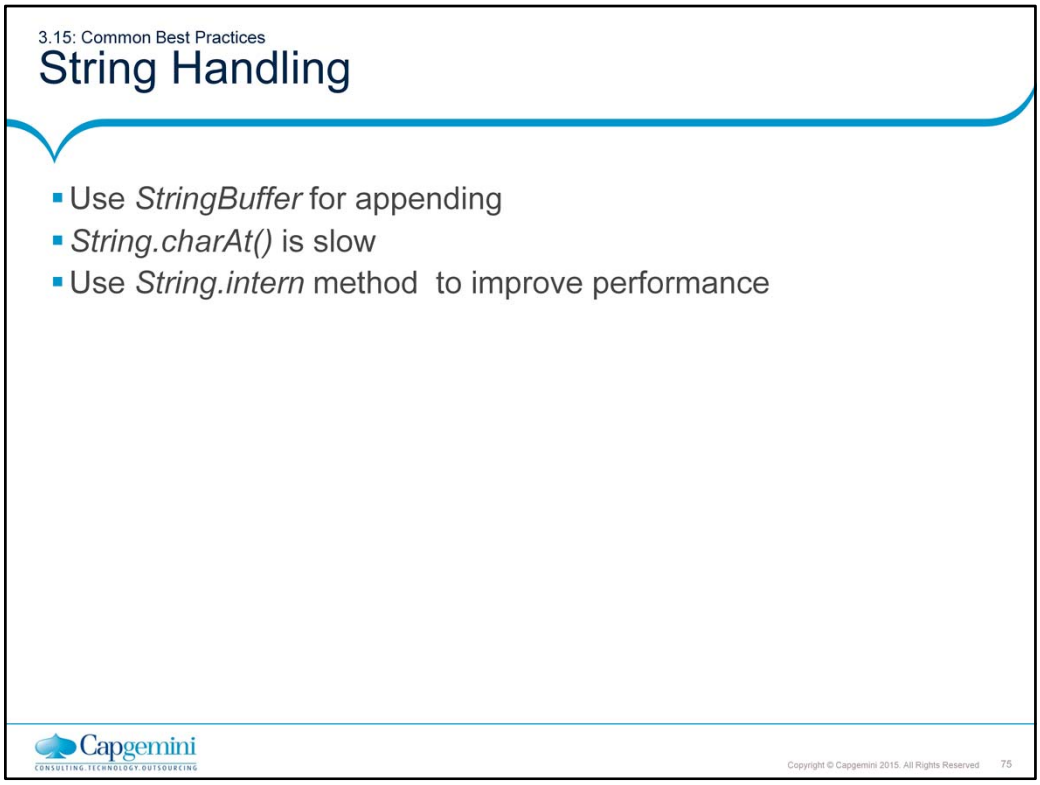
 73



Declare constants as static and final.
Use static, if variable is not going to change. This reduces the memory space for the object. A single copy is shared among all the objects.
Static, final and private methods are faster.
Static, final and private functions are treated as inline functions. These types of methods are resolved faster than other types because resolutions are done only within the class. So, if the method is not to be overridden by the sub class then define it as final, static and private.
If possible, use constants in if conditions.
Conditional Compilation may be useful. Conditional compilation gets rid of the run time resolution of variables and their values.

```
class A {
    public static final constA = 1;
}
class B {
  public static void main(String arg[])  {
       if (A.constA == 1)  System.out.println("inside if ");
  }
}
```

In the above example, during the compilation of Class B the if condition is checked. Since the condition involves the final (constant), the time required for resolving the if condition and variables are not done at runtime, which can save some time. But if the value of const "constA" is changed ensure that both classes are compiled.



Use StringBuffer for appending
Always use StringBuffer for appending. Appending by StringBuffer is almost 200 times faster than by using String.concat and "+" operator. But if String constants are to be appended "+" is faster than StringBuffer.append because it is resolved in compile time. Any code with constants is faster.

String.charAt() is slow
The method charAt(int index) returns an individual char from inside a string (see also the substring() method below). The valid index numbers are in the range 0..length-1. Using an index number outside of that range will raise a runtime exception and stop the program at that point.

```
String string = "hello"; char a = string.charAt(0); // a is 'h'
char b = string.charAt(4); // b is 'o‘
char c = string.charAt(string.length() - 1); // same as above line
char d = string.charAt(99); // ERROR, index out of bounds
```

charAt() is slow because it does check for bounds before finding the character. Secondly, it is not declared as final which actually make a bit slower. One can use indexOf and a loop. Which is at least 3 times faster than charAt().

String indexOf()
The indexOf() method searches inside the receiver string for a "target" string. The indexOf() method returns the index number where the target string is first found (searching left to right), or -1 if the target is not found.

int indexOf(String target) -- searches for the target string in the receiver. Returns the index where the target is found, or -1 if not found.

The indexOf() search is case-sensitive -- upper and lowercase letters must match exactly.

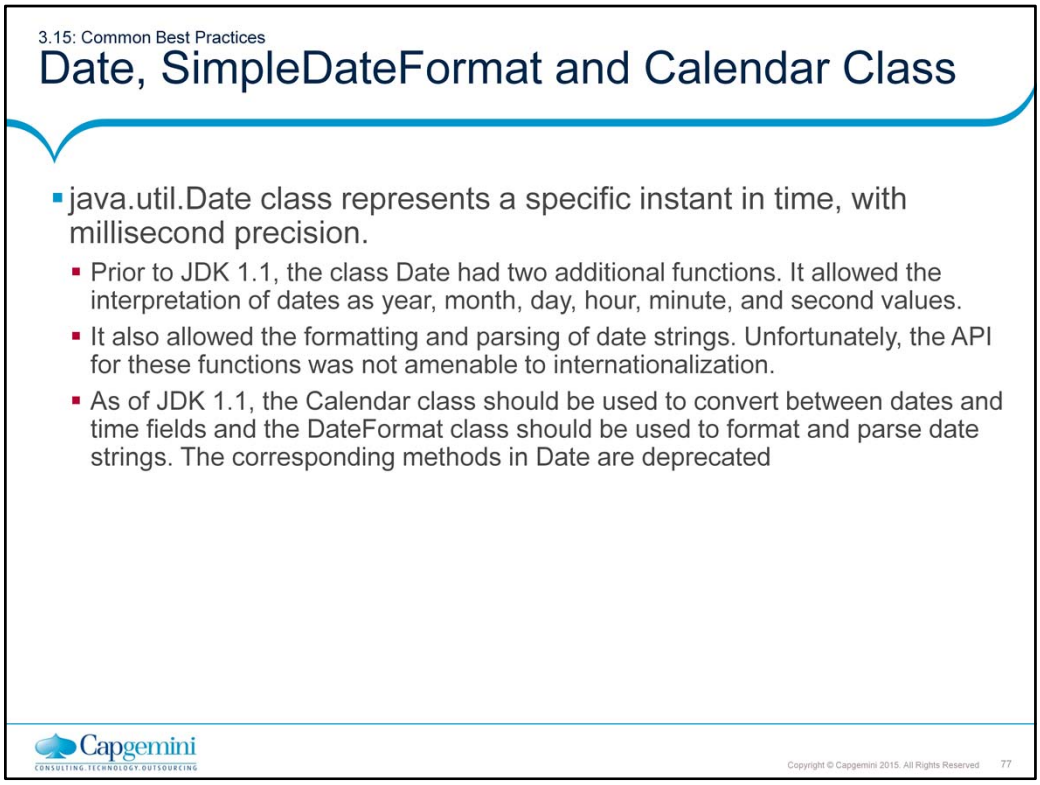
```
String string = "Here there everywhere";
int a = string.indexOf("there"); // a is 5
int b = string.indexOf("er"); // b is 1
int c = string.indexOf("eR"); // c is -1, "eR" is not found
```

String.intern method can be used to improve performance
Consider the following example:

```
for(int i=0;i<variables.length;i++){
variables[i] = new String("hello");
}

here since strings are immutable. For every assignment a
separate object is created. But if used in the following way
int len = variables.length()
for(int i=0;i<len;i++){
variables[i] = new String("hello");
variables[i] = variables[i].intern();
}
```

The problem of immutability could be solved to some extent. The calling of intern method ensures that when the value of that particular string is changed, then instead of creating new string, the reference of old string is obtained and value of that is changed. It gives considerable performance improvement. It is nothing but creating a pool of unique String objects.



Example to find difference in dates:

```
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.TimeZone;

public class DateTest {

static SimpleDateFormat sdf = new SimpleDateFormat("dd-MMM-yyyy");

public static void main(String[] args) {

  TimeZone.setDefault(TimeZone.getTimeZone("Europe/London"));

  //diff between these 2 dates should be 1
  Date d1 = new Date("01/01/2007 12:00:00");
  Date d2 = new Date("01/02/2007 12:00:00");

  //diff between these 2 dates should be 1
  Date d3 = new Date("03/24/2007 12:00:00");
  Date d4 = new Date("03/25/2007 12:00:00");

  Calendar cal1 = Calendar.getInstance();cal1.setTime(d1);
  Calendar cal2 = Calendar.getInstance();cal2.setTime(d2);
  Calendar cal3 = Calendar.getInstance();cal3.setTime(d3);
  Calendar cal4 = Calendar.getInstance();cal4.setTime(d4);
```

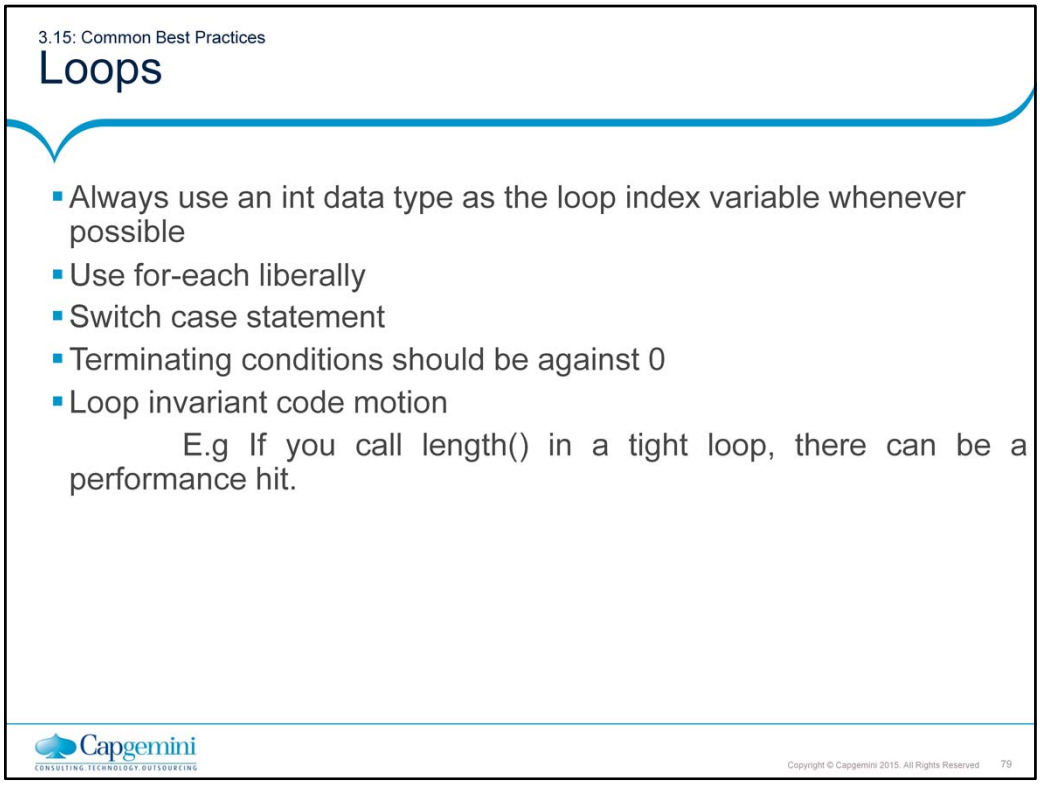
[continued on the next page]

```
printOutput("Manual   ", d1, d2, calculateDays(d1, d2));
  printOutput("Calendar ", d1, d2, daysBetween(cal1, cal2));
  printOutput("Manual   ", d3, d4, calculateDays(d3, d4));
  printOutput("Calendar ", d3, d4, daysBetween(cal3, cal4));
}


private static void printOutput(String type, Date d1, Date d2, long result) {
  System.out.println(type+ "- Days between: " + sdf.format(d1)
                  + " and " + sdf.format(d2) + " is: " + result);
}

/** Manual Method - YIELDS INCORRECT RESULTS - DO NOT USE**/
/* This method is used to find the no of days between the given dates */
public static long calculateDays(Date dateEarly, Date dateLater) {
  return (dateLater.getTime() - dateEarly.getTime()) / (24 * 60 * 60 * 1000);
}

/** Using Calendar - THE CORRECT WAY**/
public static long daysBetween(Calendar startDate, Calendar endDate) {
  Calendar date = (Calendar) startDate.clone();
  long daysBetween = 0;
  while (date.before(endDate)) {
    date.add(Calendar.DAY_OF_MONTH, 1);
    daysBetween++;
  }
  return daysBetween;
}
}
```



Always use an int data type as the loop index variable whenever possible.
It is efficient when compared to using byte or short data types. This is because when we use byte or short data type as the loop index variable they involve implicit type cast to int data type.

Use for-each liberally.
The for-each loop is used with both collections and arrays. It is intended to simplify the most common form of iteration, where the iterator or index is used solely for iteration, and not for any other kind of operation, such as removing or editing an item in the collection or array.

When there is a choice, the for-each loop should be preferred over the for loop, since it increases legibility.

Switch Case statement.
One peculiar fact about switch statements is that code consisting of case with consecutive constants like 0,1,2 are faster than with case with constants like 2, 6, 7, 14, etc. This is because in the former switching between options requires less offset and it takes only 16 bytes. But in the later the offset is higher and it might take 32 or more bytes.

Terminating conditions should be against 0.
For example:

```
for(int i=0;i<a.length;i++){
     a[i]+=i; }

for(int i=a.length-1;i>=0;i--){
     a[i]+=i; }
```

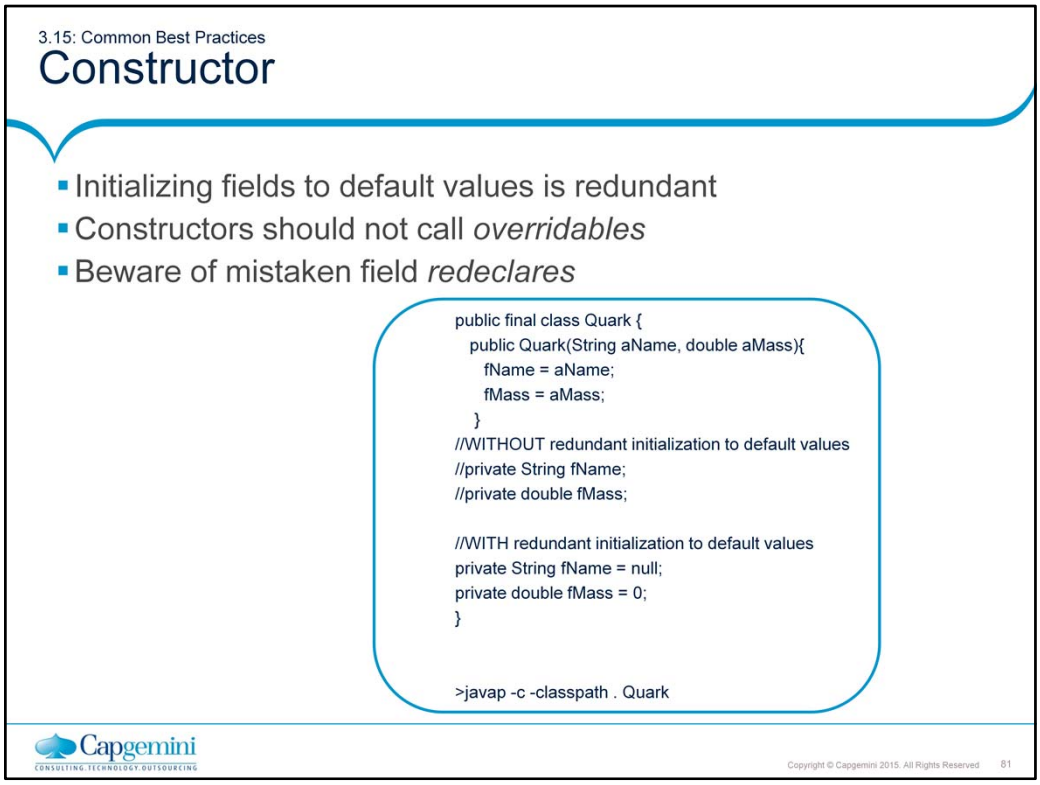
The later one is 30% faster than the former. Just you are comparing with constant instead of accessing a.length everytime.

Loop invariant code motion.
If in a loop a expression is evaluated in every iteration the performance may be slow. This is called "Loop invariant code motion". Instead evaluate the expression outside the loop.
length() is a method in String used to obtain the number of characters currently in the string. If you call length()in a tight loop, there can be a performance hit.

Example:

```
for(int i =0;i <s.length(); i++) {    //AVOID
…
}

// precomputed length
          int len = s.length();
for(int i =0; i < len; i++) {
….
}
```

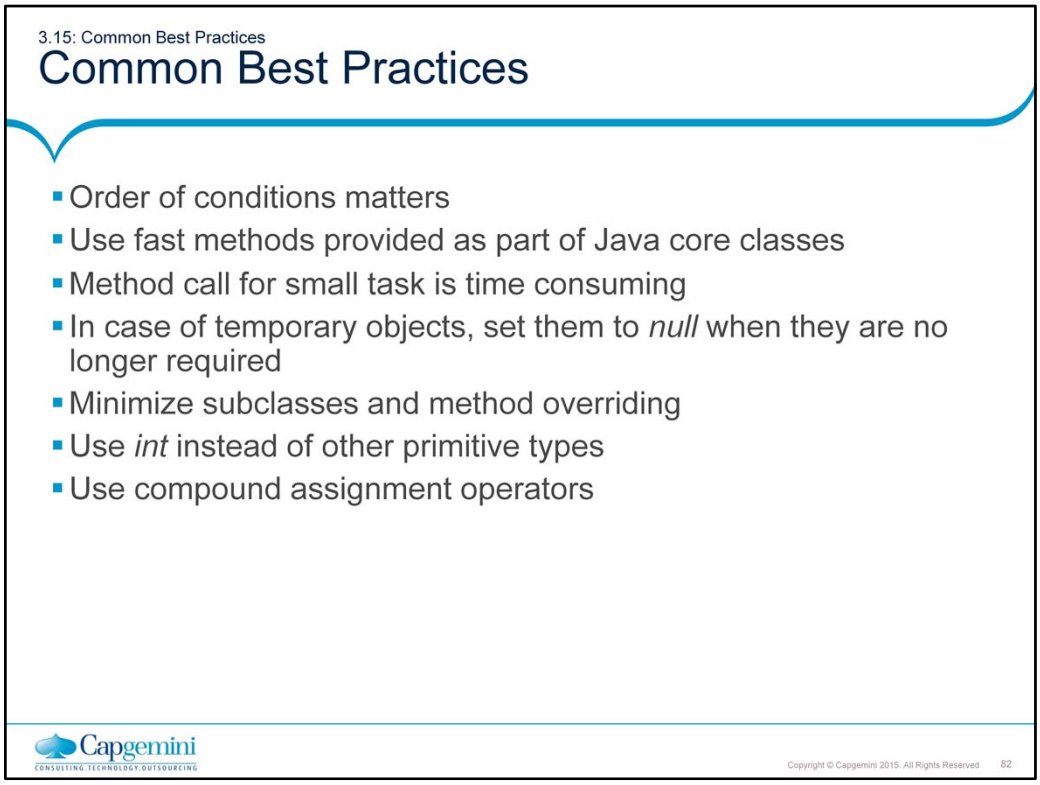


Initializing fields to default values is redundant.
In the declaration of a field, setting it explicitly to its default initial value is always redundant, and may even cause the same operation to be performed twice (depending on your compiler). Declaring and initializing object fields as null, for instance, is simply not necessary in Java.

(It is worth recalling here that Java defines default initial values only for *fields*, and *not* for local variables).

Constructors should not call overridables.
That is, they should only call methods that are private, static, or final.

Beware of mistaken field re-declares.
Beware of this simple mistake, which can be hard to track down : if a field is mistakenly redeclared within the body of a method, then the field is not being referenced, but rather a temporary local variable with the same name. A common symptom of this problem is that a stack trace indicates that a field is null, but a cursory examination of the code makes it seem as if the field has been correctly initialized.



Order of conditions matters
Put the condition which has more probability of being true to the left most part of the compound condition.
Use Fast Methods Provided with Java's Core Classes
It is common to need to copy values or object references from one array to another, and you might do so using the following code:

```
 int[] firstArray, secondArray;

// ...

for (int i = 0; i < firstArray.length; i++) {

  secondArray[i] = firstArray[i];
}
```

However, a much faster way to accomplish the same result is to use the arraycopy() method defined in the System class as shown below:

```
int[] firstArray, secondArray;
// ...
System.arraycopy(firstArray, 0, secondArray, 0, firstArray.length);
```

Method call for small task is time consuming

Method call may be time consuming, especially if method does a very small job (like finding maximum of two numbers). The method call for such thing is costly because runtime resolution of the methods. The JVM needs to resolve the correct class of which the method is called. This is more a issue if class's hierarchy is long one.

For a small thing do not write a method, it is wiser to write inline code. If this is not possible then in the case of long hierarchy try specifying final to the method so that it resolved in the compile time itself.

In case of temporary objects, set it to null when no longer required.

Java's garbage collection process is very complex. Moreover it is not fixed when it is going to be executed. Typically the gc is executed it checks the objects in the java heap which contains all objects instantiated. It finds whether the object is referred somewhere or not. If it is not then it executes the finalize method for that class and then places it in the gc heap and discarded. The process of detecting whether if it is used or not is very costly.

This is compounded if the large number of objects are made in the program. One method to reduce this is to assign null value to the object if it is not used. This reduces the checking by GC and enables it to directly place it in the GC heap.

Minimize subclasses and method overriding

To improve performance

Use int instead of other primitive types

Operations performed on int primitives generally execute faster than for any other primitive type supported by Java, so you should use int values whenever possible; char and short values are promoted to int automatically before arithmetic operations.

Use compound assignment operators

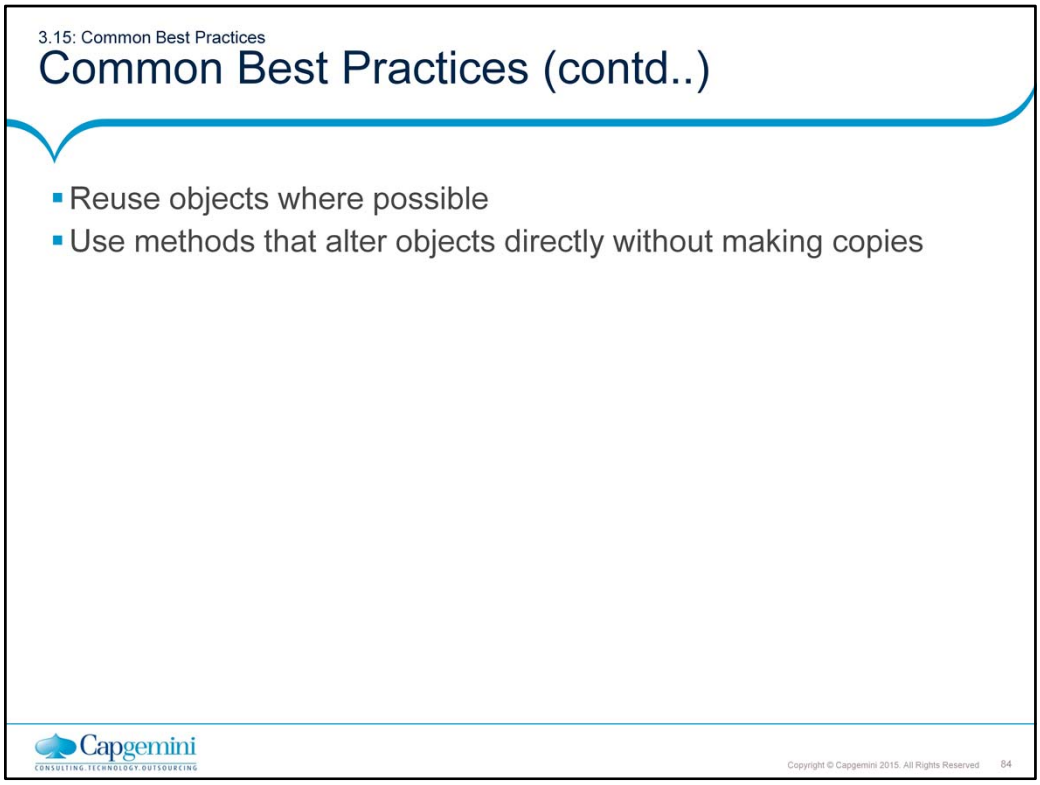
You might expect that a += b; is identical to a = a + b; when the two are compiled, but that is not the case. In fact, these cause different Java byte codes to be generated, and the second approach actually takes longer to execute. Compound assignment operators such as += are compiled into less code than the equivalent mathematical and assignment operators. Therefore, you can make minor improvements in the speed of your code by using compound operators such as +=, -=, *=, and /=.

Compound assignment statements first perform an operation on an argument before assigning it to another argument. Example: a += b

In the example above, b is added to the value of a, and that new value is then assigned to a. Compound assignment operators are more concise than their constituent operators (separate + and =).

Compare the statement above to the one below. The statement above is a shorthand equivalent of the statement below. Example: a = a + b

Compound assignment operators are faster. If you are operating on an expression instead of a single variable, for example an array element, you can achieve a significant improvement with compound assignment operators.

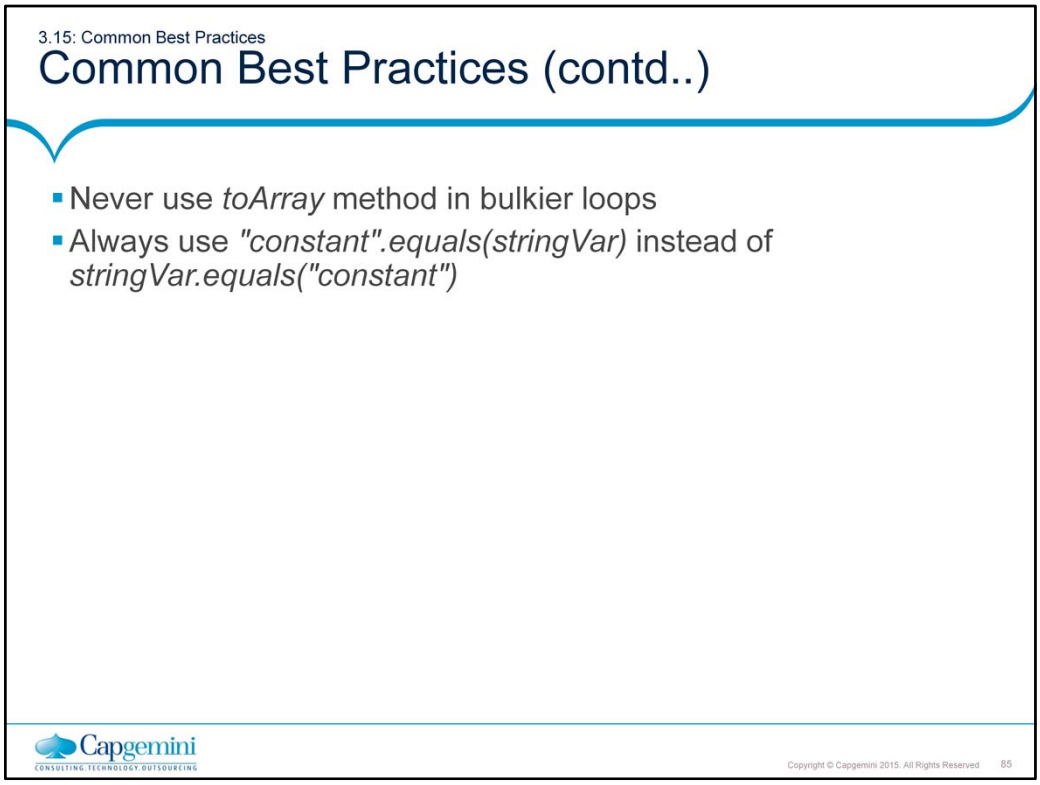


Reuse objects where possible
Avoid creating objects in frequently used routines. Because these routines are called frequently, you will likely be creating objects frequently, and consequently adding heavily to the overall burden of object cycling. By rewriting such routines to avoid creating objects, possibly by passing in reusable objects as parameters, you can decrease object cycling.

Empty collection objects before reusing them. (Do not shrink them unless they are very large).

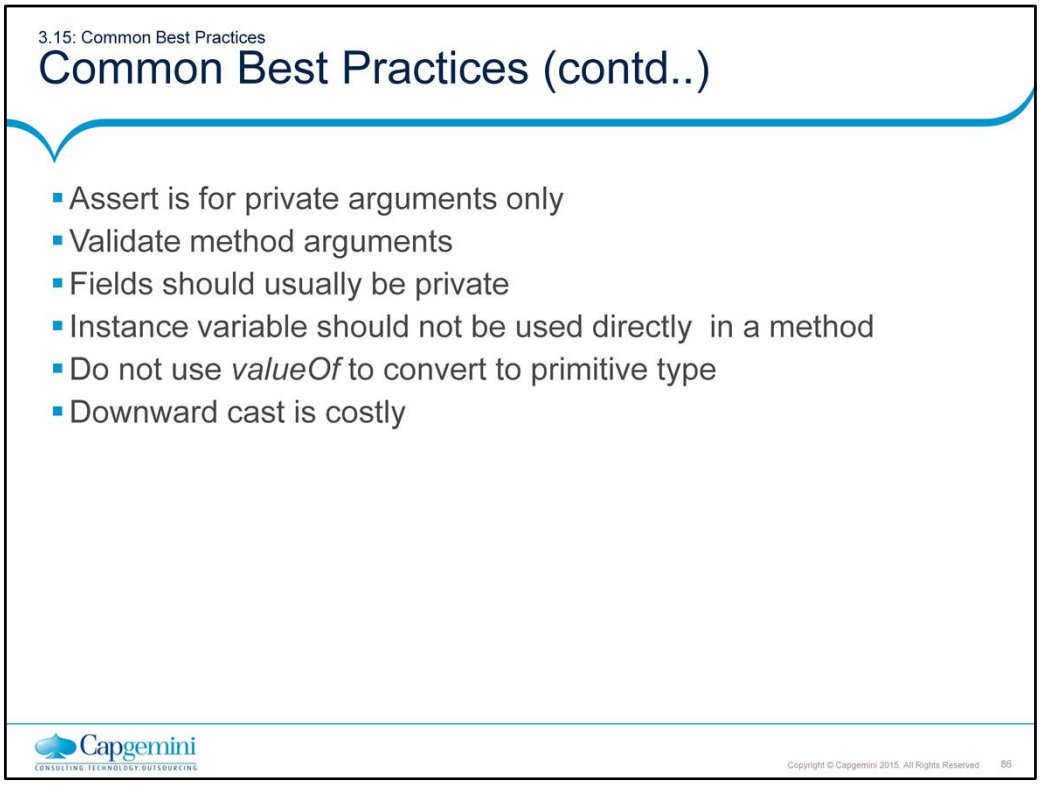
Most container objects (e.g., Vectors, Hashtables) can be reused rather than created and thrown away. Of course, while you are not using the retained objects, you are holding on to more memory than if you simply discarded those objects, and this reduces the memory available to create other objects. You need to balance the need to have some free memory available against the need to improve performance by reusing objects. But generally, the space taken by retaining objects for later reuse is significant only for very large collections, and you should certainly know which ones these are in your application.

Use methods that alter objects directly without making copies
Instead of making local copy of object, directly modify the original object.



Never build arrays from toArray methods of Collection classes like HashMap and ArrayList in a bulkier loop. It will halt the system.

Always use "constant".equals(stringVar) instead of stringVar.equals("constant"). It gets rid of possible NULL pointer exception.



Assert is for private arguments only

Most validity checks in a program are checks on parameters passed to non-private methods. The assert keyword is not meant for these types of validations.

Assertions can be disabled. Since checks on parameters to non-private methods implement the requirements demanded of the caller, turning off such checks at runtime would mean that part of the contract is no longer being enforced.

Conversely, checks on arguments to private methods can indeed use assert. These checks are made to verify assumptions about internal implementation details, and not to check that the caller has followed the requirements of a non-private method's contract.

Validate method arguments

The first lines of a method are usually devoted to checking the validity of method arguments. The idea is to fail as quickly as possible in the event of an error. This is particularly important for constructors. It is a reasonable policy for a class to skip validating arguments of private methods. The reason is that private methods can only be called from the class itself. Thus, a class author should be able to confirm that all calls of a private method are valid. If desired, the assert keyword can be used to check private method arguments, to check the internal consistency of the class.

Fields should usually be private:
Fields should be declared private unless there is a good reason for not doing so.
One of the guiding principles of lasting value in programming is "Minimize ripple effects by keeping secrets." When a field is private, the caller cannot usually get inappropriate direct access to the field.

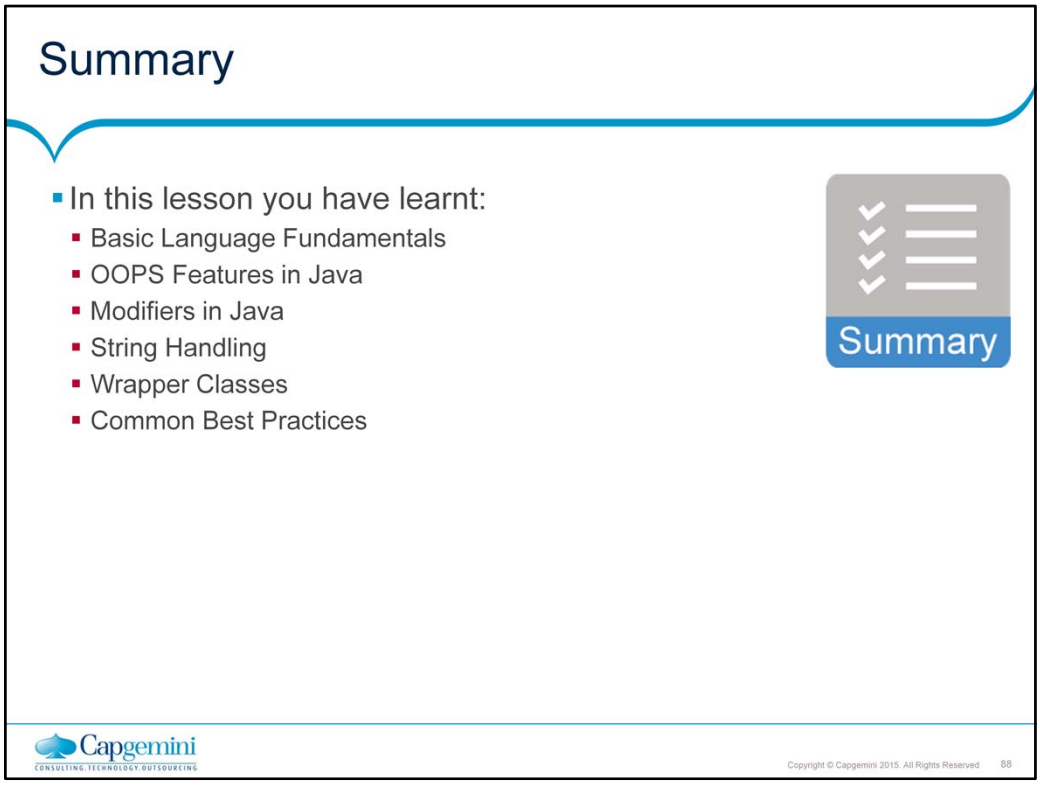
Accessing Instance variables in a method:

```
private char buf[] = new char[4096];
public int find(char c)
{ for (int i = 0; i < buf.length; i++) {
if (buf[i] == c)
return i; }
return -1;}
```

The above code is optimized drastically if the instance variable buf is copied to a local variable in the method and used in the loop. This is because the JVM invokes getField method internally to access the instance variable which is a overhead. Another optimization is never use ".length" of array in the for loop. Instead get the length in a local variable and use that in for loop condition, because JVM calls arrayLength method internally for every iteration

ValueOf (double) to convert to primitive type
Avoid using this method. It creates a unnecessary Double object inside this method. Instead use parseDouble. Follow the same for other wrapper classes also

Downward cast is costly
Downward cast from parent class to its subclass/es are always costly, since it has to check the validity of both the classes and if not valid it has throw an exception "ClassCastException" which is time consuming. Instead first check the validity of subclass using **instanceOf** and then cast it. It removes the chance of throwing exception



Add the notes here.

Review Questions

- Question 1: Which of the following is true regarding enum?
  - **Option1:** enum cannot be used inside methods.
  - **Option2:** enum need not have a semicolon at the end.
  - **Option3:** enum can be only declared with public or default access specifier.
  - **Option4:** All the above are true.

- Question 2: Java considers variable number and NuMbEr to be identical.
  - True/False

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 89

Review Questions

- Question 3: The *do...while* statement tests the loop-continuation condition _____ it executes executing the loop's body; hence, the body executes at least once.
  - **Option1:** before
  - **Option2:** after

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Introduction to Java

Lab Book

Document Revision History

Date	Revision No.	Author	Summary of Changes

Table of Contents

Getting Started

Overview

This lab book is a guided tour for learning Core Java version 5.0. It comprises solved examples and 'To Do' assignments. Follow the steps provided in the solved examples and work out the 'To Do' assignments given which will expose you to working with Java applications. Creating Classes, Interfaces, GUI-based applications and applets, Database application, networking and mailing concepts and logging in.

Setup Checklist for Core Java

Here is what is expected on your machine in order for the lab to work.

Minimum System Requirements

- Intel Pentium 90 or higher (P166 recommended)
- Microsoft Windows 95, 98, or NT 4.0, 2k, XP.
- Memory: 32MB of RAM (64MB or more recommended)
- Internet Explorer 6.0 or higher
- MS-Access/Connectivity to Oracle database
- Apache Tomcat Version 5.0.

Please ensure that the following is done:

- A text editor like Notepad or Eclipse is installed.
- JDK 1.5 is installed. (This path is henceforth referred as <java_install_dir>)

Instructions

- For all coding standards refer Appendix A. All lab assignments should refer coding standards.
- Create a directory by your name in drive <drive>. In this directory, create a subdirectory java_assgn. For each lab exercise create a directory as lab <lab number>.

Learning More (Bibliography if applicable)

- Java Programming Advanced Topics: Source Book by Wigglsworth and McMilan
- Java: How to Program by Dietel n Dietel
- Java2: Beginning Java2 JDK 5 by Doug Lowe,Joel Murach and Andrea Steelman.

Lab 1. Working with Eclipse

Goals	• Learn and understand the process of: ○ Setting environment variables ○ Creating a simple Java Project using Eclipse 3.0
Time	20 minutes

1.1: Setting environment variables from CommandLine.Solution:

Step 1: Set **JAVA_HOME** to Jdk1.5 using the following command:

- **Set JAVA_HOME=C:**\Program Files\Java\Jdk1.5.0.07

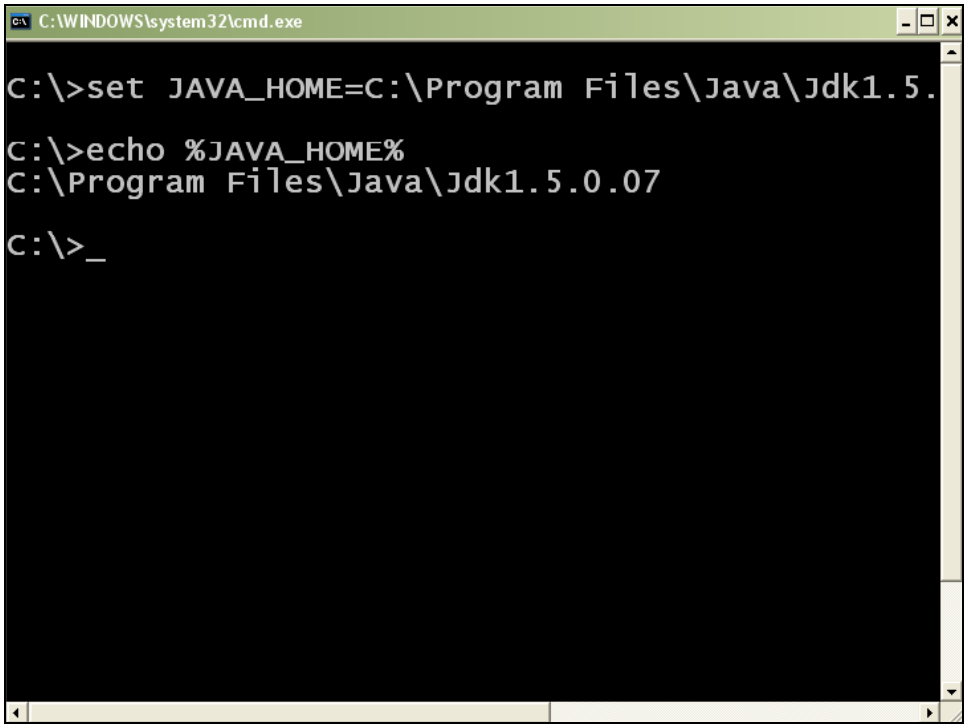


Figure 1: Java program

Step 2: Set PATH environment variable:

- **Set PATH=%PATH%;%JAVA_HOME%\bin;**

Step 3: Set your current working directory and set classpath.

- Set CLASSPATH=.

Note: Classpath searches for the classes required to execute the command. Hence it must be set to the directory containing the class files or the names of the jars delimited by ;

For example: C:\Test\myproject\Class;ant.jar

Alternatively follow the following steps for setting the environment variables

Alternate approach:

Step 1: Right click **My Computers**, and select **Properties → Environment Variables**.

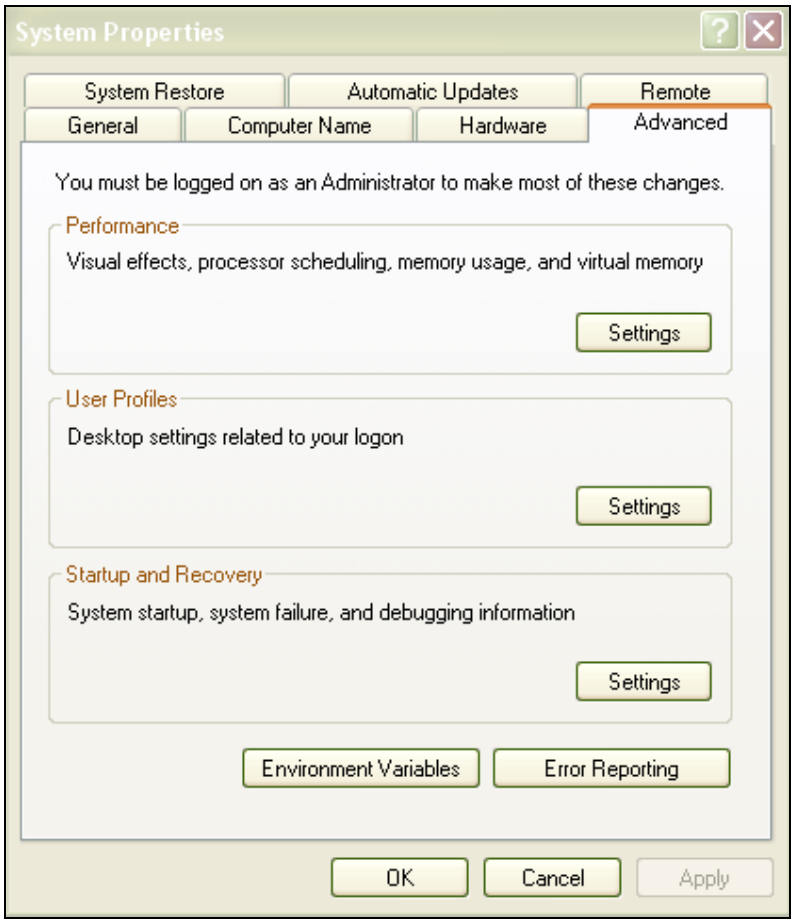


Figure 2: System Properties

Step 2: Click **Environment Variables**. The Environment Variables window will be displayed.

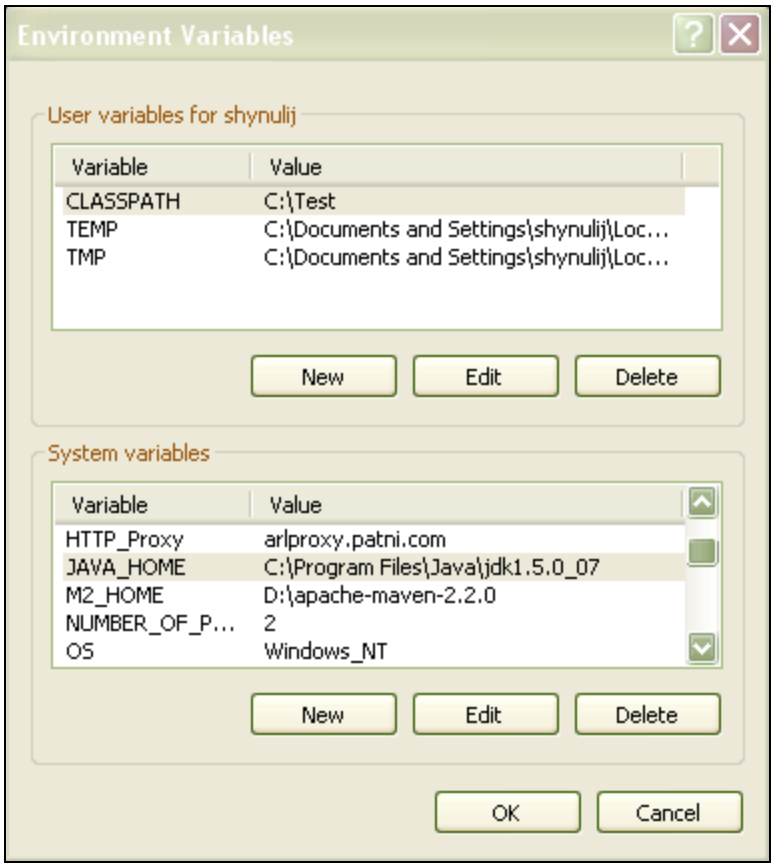


Figure 3: Environment Variables

Step 3: Click **JAVA_HOME** System Variable if it already exists, or create a new one and set the path of JDK1.5 as shown in the figure.

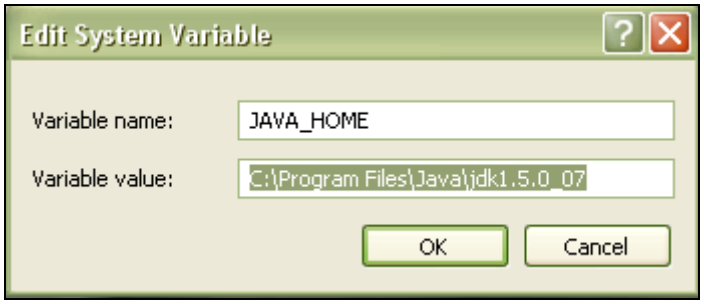


Figure 4: Edit System Variable

Step 4: Click **PATH** System Variable and set it as ***%PATH%;%JAVA_HOME%\bin***.

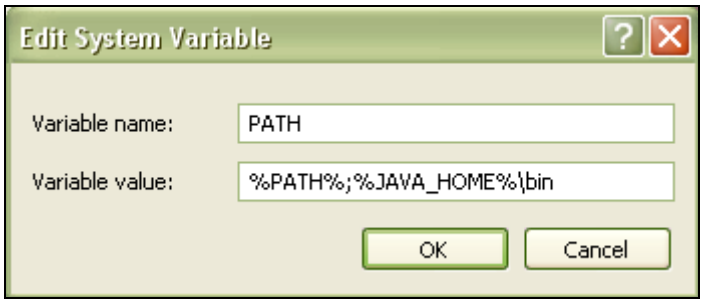


Figure 5: Edit System Variable

Step 5: Set **CLASSPATH** to your working directory in the **User Variables** tab.

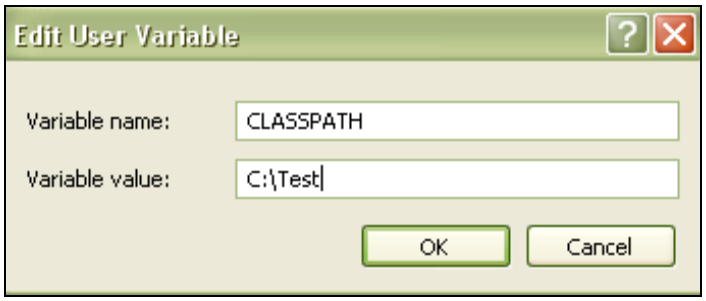


Figure 6: Edit User Variable

1.2: Create Java Project

Create a simple java project named ‘myproject’..

Solution:

Step 1: Open **eclipse3.3**.

Step 2: Select **File → New → Project → Java project**.

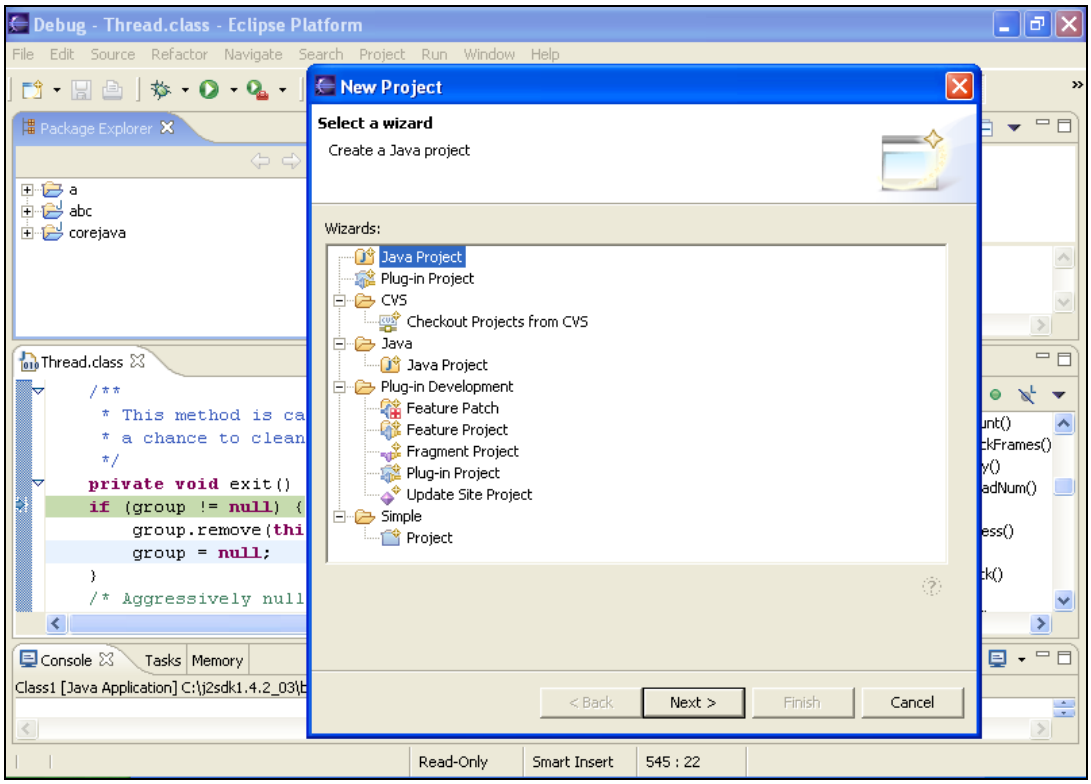


Figure 7: Select Wizard

Step 3: Click **Next** and provide name for the project.

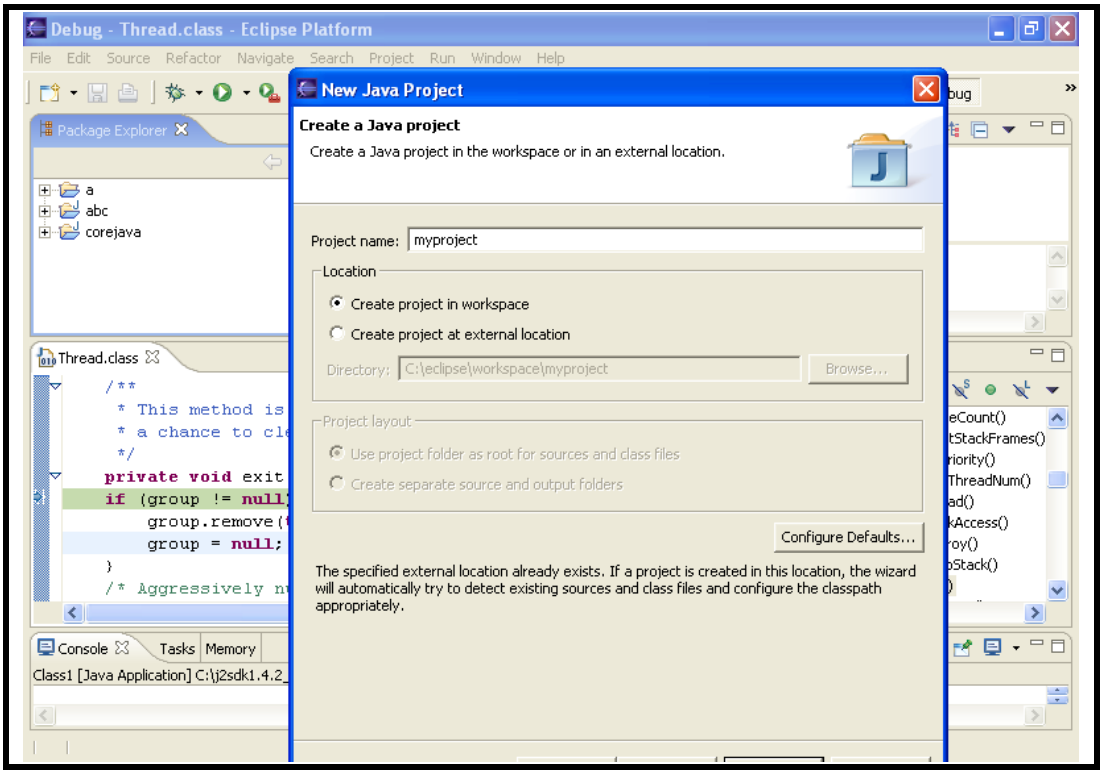


Figure 8: New Java Project

Step 4: Click **Next** and select build options for the project.

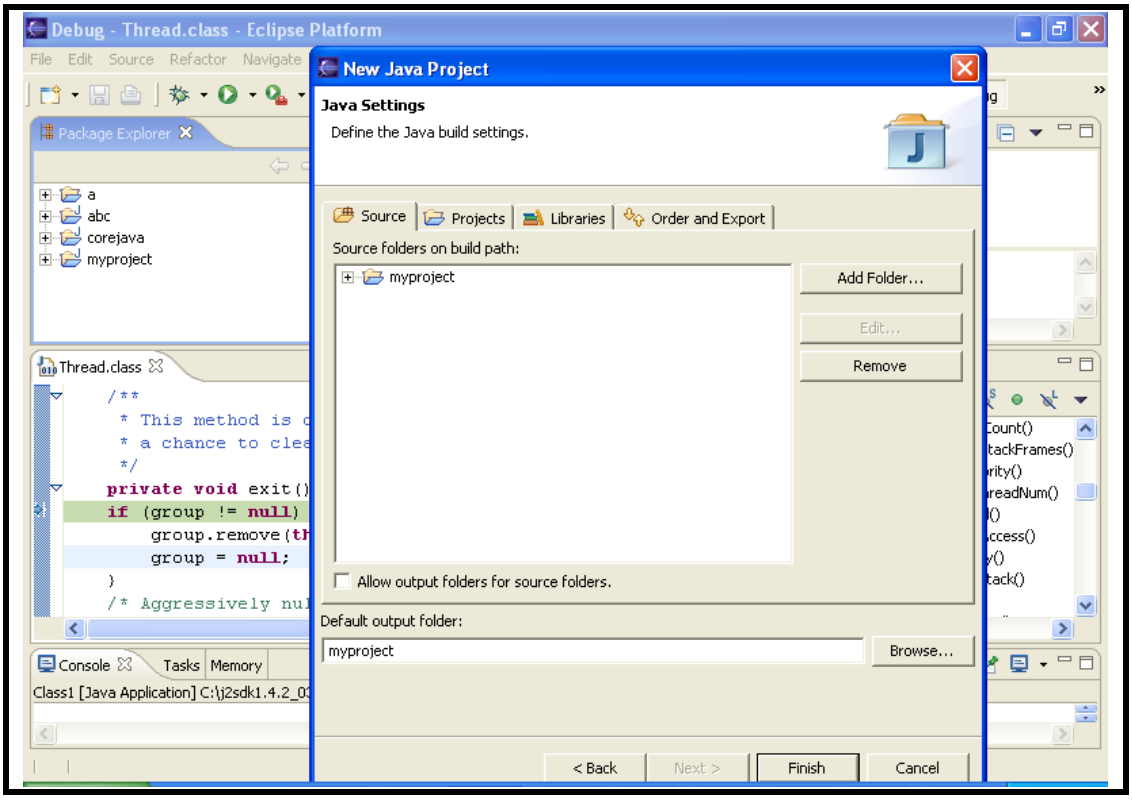


Figure 9: Java Settings

Step 5: Click **Finish** to complete the project creation.

Step 6: Right-click **myproject**, and select resource type that has to be created.

For example: Class if it is a Java class

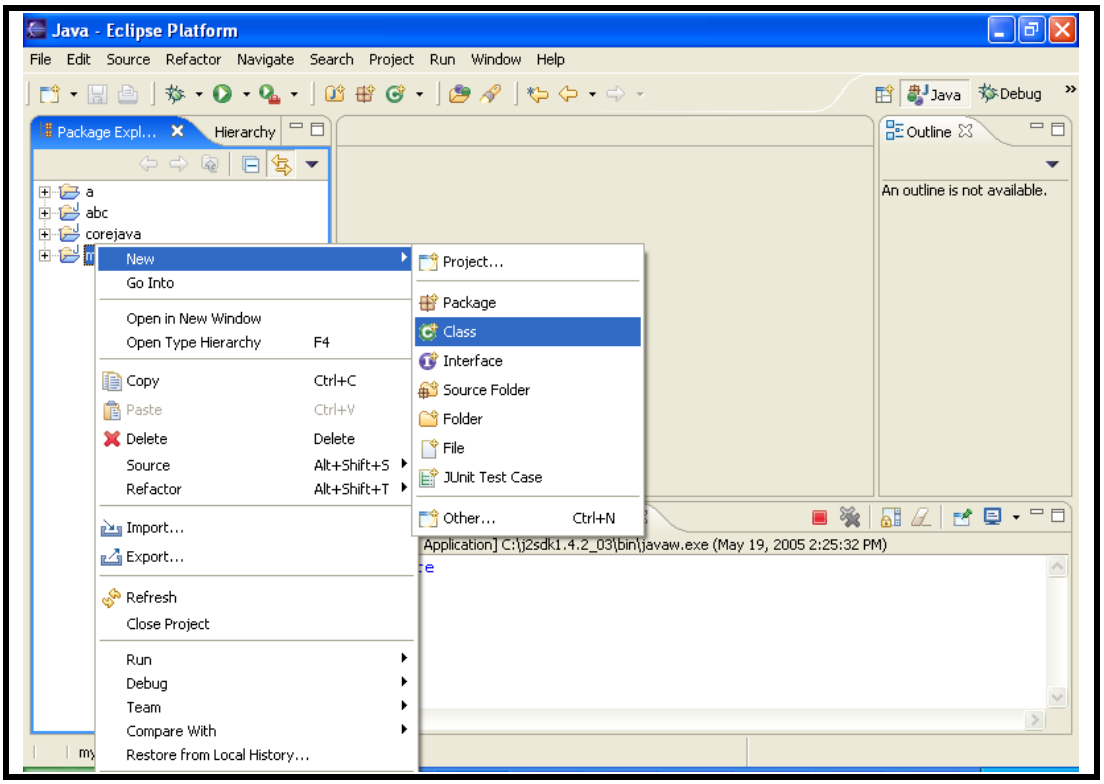


Figure 10: Select Resource

Step 7: Provide name and other details for the class, and click **Finish**.

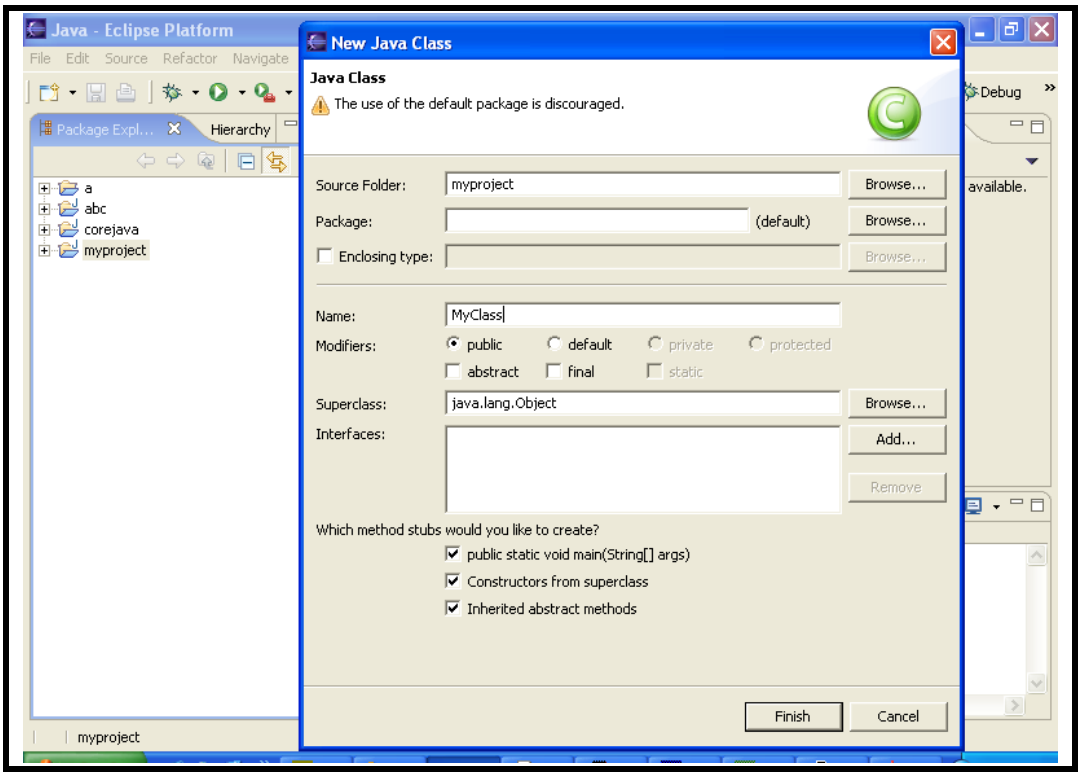


Figure 11: Java Class

This will open **MyClass.java** in the editor, with ready skeleton for the class, default constructor, **main()** method, and necessary **javadoc** comments.

To run this class, select **Run** from toolbar, or select **Run As → Java application**. Alternatively, you can select **Run..** and you will be guided through a wizard, for the selection of class containing **main()** method.

Console window will show the output.

Lab 2. Language Fundamentals – Part I

Goals	• At the end of this lab session, you will be able to: ○ Write a Java program that diplays Hello world ○ Working with Conditional Statements ○ Create Classes and Objects
Time	1hr 30 min

2.1: Write a program that displays “Hello World”

Solution:

Step 1: Right click the Project **myproject**, and select **New → Class**.

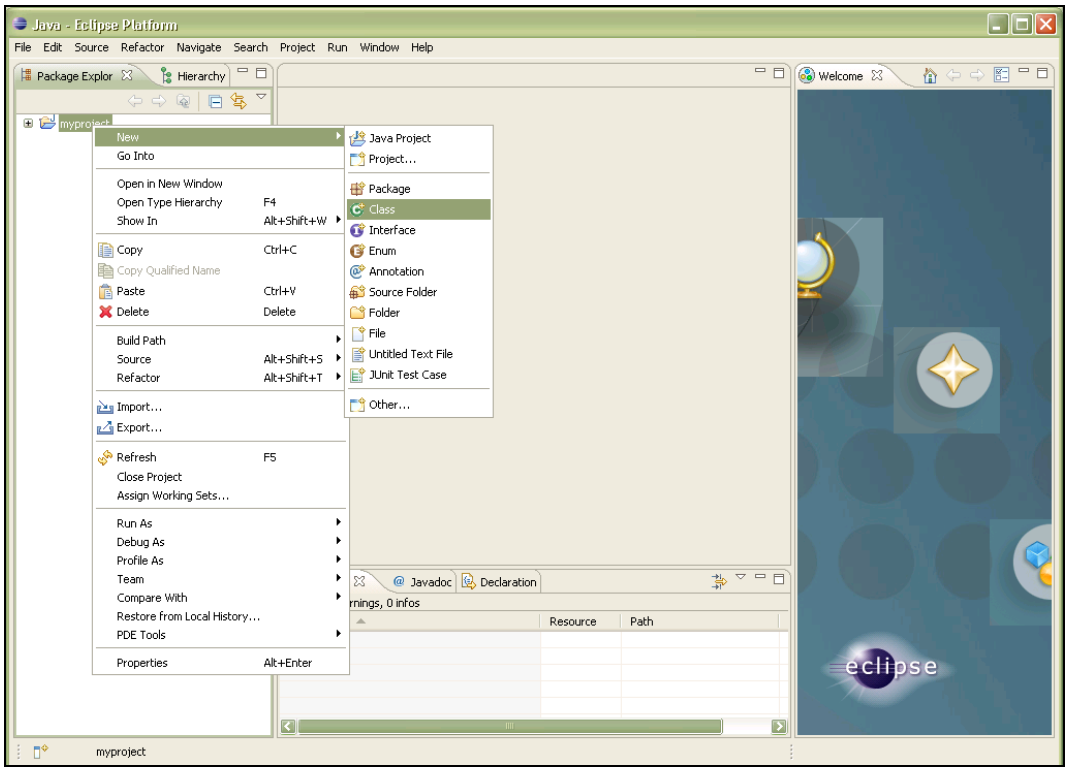


Figure 12: Creating a New Class

Step 2: In the **New Java Class** dialog, key in the name of the class as "***HelloWorld***".

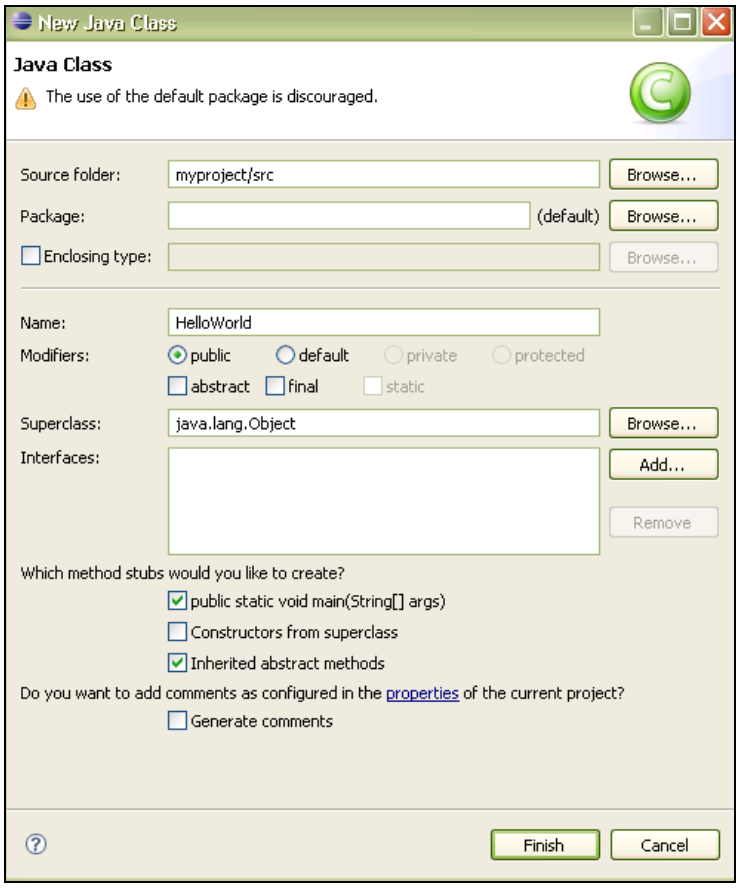


Figure 13: New Java Class dialog box

Step 3: Click **Finish**. The code–window will be displayed before you**.**

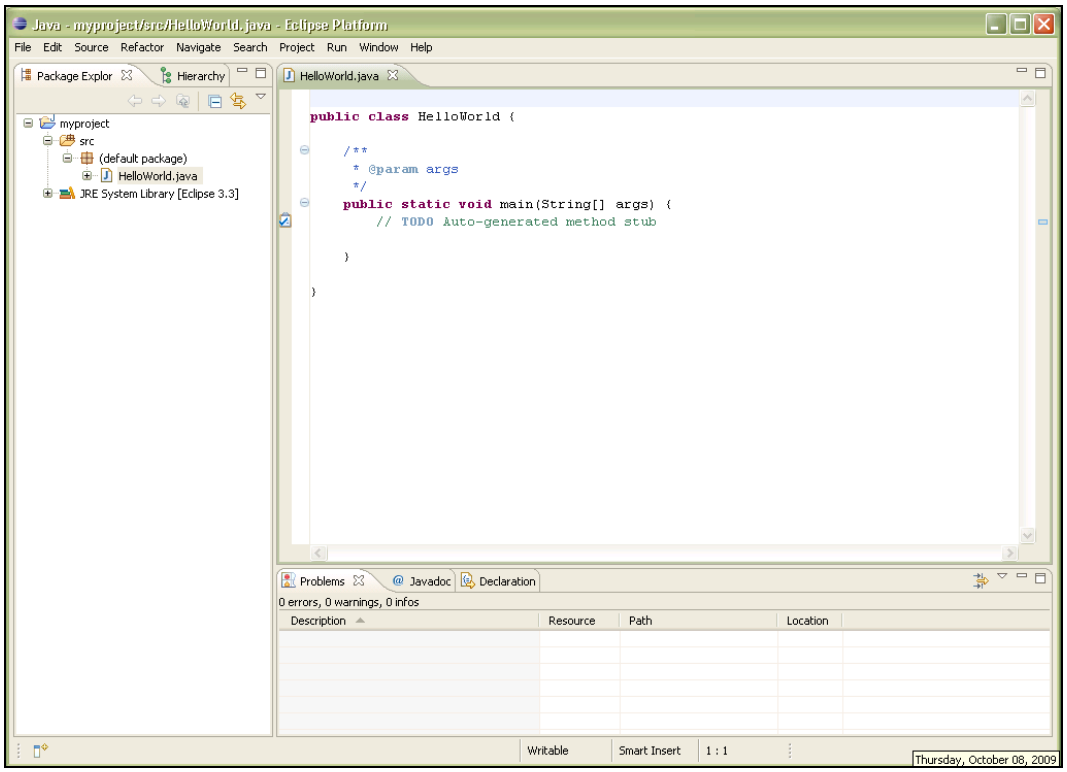


Figure 14: Code window

Step 4: Type the following code in the window and save the changes.

```
class HelloWorld {

        /*This is a block comment */

        public static void main(String args[]) {

                System.out.println("Hello World");

        } //main ends.

} //class ends
```

Example 1: HelloWorld.java

Step 5: Run the code by right clicking in the code window, and selecting **Run As → Java Application**.

Alternatively, from the main menu, select **Run → Run As → Java Application**.

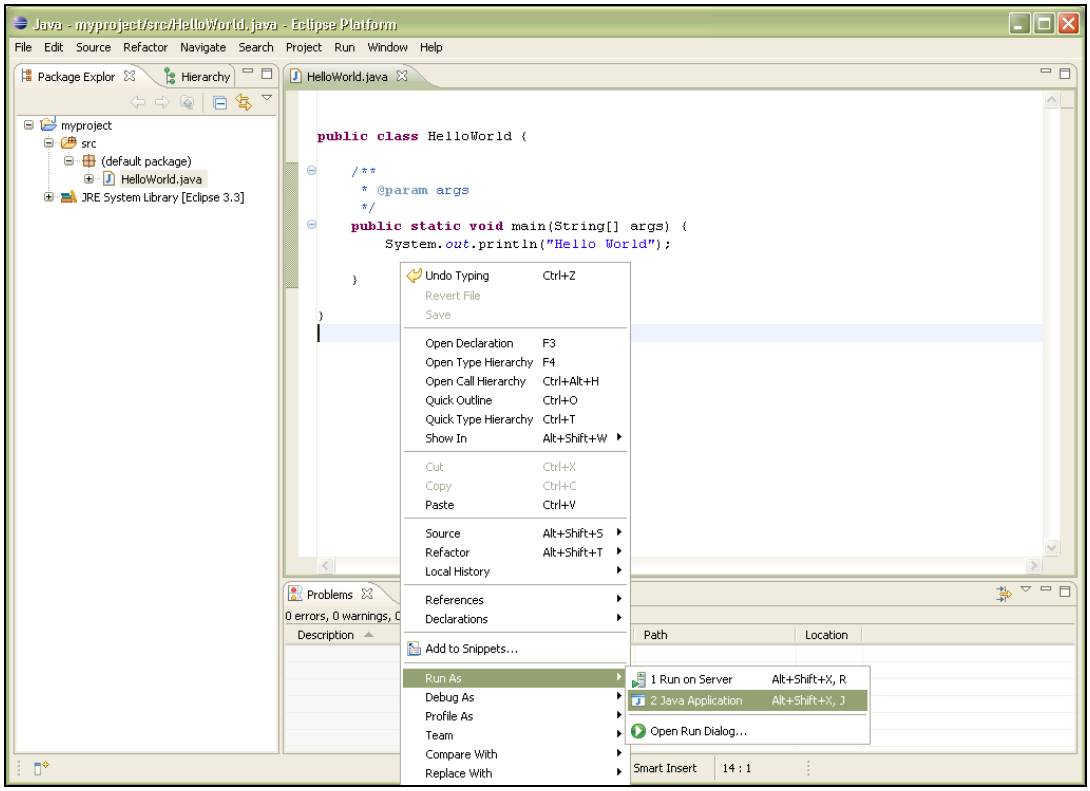


Figure 15: Selecting the Java Application

Step 6: The code will be executed and the output will be produced in the console window.

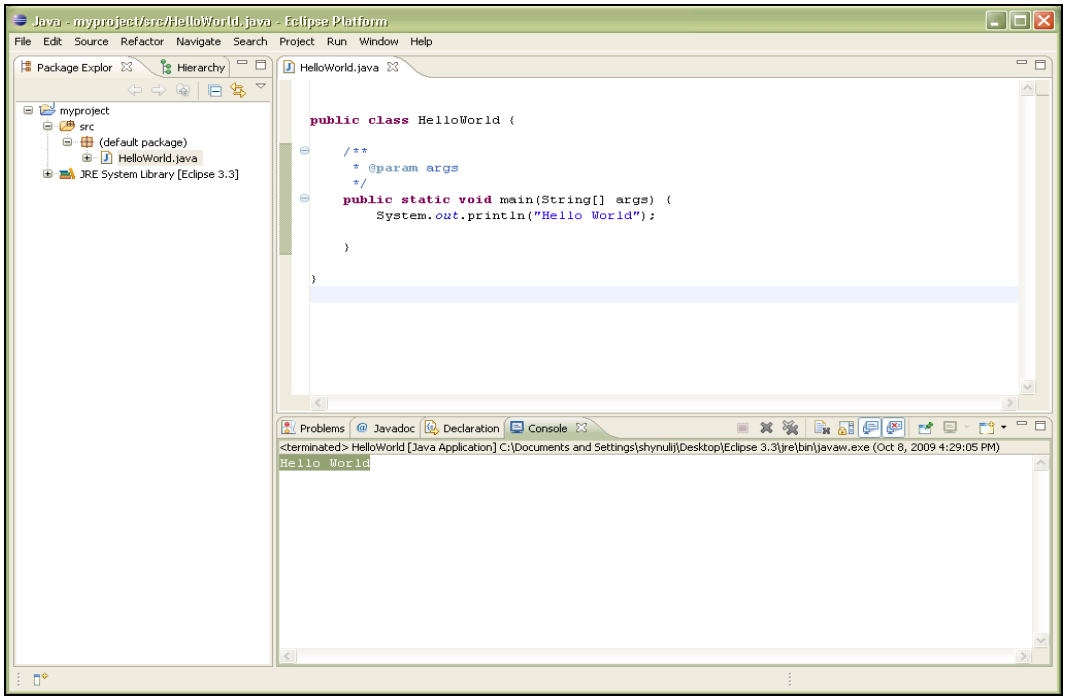


Figure 16: Console window

Alternatively to execute the same on the command prompt follow the steps given below:

Step 1: Open a notepad and type the code for HelloWorld.java.

Step 2: Open the command prompt and set your working directory.

Step 3: Compile the program by typing the following command:
Command: javac HelloWorld.java

Step 4: After successful compilation let us now execute the program using the following command:
Command: java HelloWorld

Step 5: You should get the output as shown below:
Hello World

<<TO DO>>

Assignment-1: Modify the **HelloWorld.java** program. Accept an integer value at command line and print the message that many number of times.

For example: c:\>HelloWorld 2 should print message 2 times.

(**Hint:** Use the **args[]** parameter of **main()**)

2.2: Working with Classes and Objects

Write a program to create a Date class and UseDate class which instantiates the Date class.

Solution:

Step 1: Type the following code in a text editor and editor and save the file with the name "**UseDate.java**".

```
class Date {
        int intDay, intMonth, intYear;

        Date(int intDay, int intMonth, int intYear)  // Constructor
        {
                this.intDay = intDay;
           this.intMonth = intMonth;
           this.intYear = intYear;
        }

        void setDay(int intDay)     // setter method for day
        {
                this.intDay = intDay;
        }

        int getDay( )                   // getter method for month
        {
                return  this.intDay;
        }

        void setMonth(int intMonth)
        {
                this.intMonth = intMonth;
        }

        int getMonth( )
```



```
        {
                return  this.intMonth;
        }

        void setYear(int intYear)
        {
                this.intYear=intYear;
        }

        int getYear( )
        {
                return  this.intYear;
        }


public String toString() //converts date obj to string.
        {
                return    "Date is "+intDay+"/"+intMonth+"/"+intYear;
        }


} // Date class

class UseDate
{
        public static void main(String[] args)
        {
                Date d = new Date(9,5,2011);
                System.out.println(d);  //invokes toString() method
        }
} //class ends
```

Example 2: UseDate.java

Step 2: Execute the class**.** You should get the output as follows:
Output: Date is 9/5/2011

<<TO DO>>

Assignment-3: Implement the following in the above date class

a. Write methods to incorporate validation checks. (For example: day should be between 1 & 31, Year must be less than 1984).

b. Call the validate methods in main. If the validation fails, restore default date.

Assignment-4: Create a class **Staff** that has members as specified below. Implement the **setter** and **getter** methods for the data members. Make the changes as required and save the file as **Staff.java**.

```
public class Staff
{
	private int staffCode;	// Employee Code
	private String staffName;		// Name
	private String designation;		// Designation
	private int age;		// Age
	private Date dateOfBirth;		// Date of Birth (Create an object of the Date
class given in the Example 2.2 or make use of Calender object)

	public Staff()	// Constructor
	{
		// To Do: Initialize data members
	}

	public Staff(int staffCode, String staffName)
	{
		// To Do: Initialize data members based on the paramters
	}

	public Staff(int staffCode, String staffName,String designation, int age)
	{
		// To Do: Initialize data members based on the parameters
	}

	// setter methods
	void setStaffCode(int staffCode)
	{			// To Do: Setter method for staff code
	}

	public void setStaffName(String staffName)
	{
		// To Do: Setter method for staffName
	}

	void setDesignation(String  designation)
	{
		// To Do: Setter method for designation
	}

	void setAge(int age)
	{
		// To Do: Setter method for age
	}
```



```
        void setDateOfBirth(Date birthdate)
        {
                // To Do: Setter method for date of birth
        }


        // getter methods
        int getStaffCode()
        {
                // To Do: getter method for staffCode
        }

        String getStaffName()
        {
                // To Do: getter method for staffName
        }

        String getDesignation()
        {
                // To Do: getter method for designation
        }

        int getAge()
        {
                // To Do: getter method for Age
        }

        Date  getDateOfBirth()
        {
                // To Do: getter method for Salary
        }

        // prints the staff details on the screen
        public void displayDetails()
        {
                System.out.println("The staff code  is "+ staffCode);
                // To Do: Print the other members in the same format
        }

} // end of class Employee
Class  StaffApplication
{
        public static void main(String[] args)
        {
                Staff staff = new Staff( );
                staff.displayDetails( );
        }
}
```


Example 3: Sample code

Hint : click generate setter and getter from source menu for setter and getter methods

Now, override the **toString()** method of **Object** class to return employeeCode and employeeName.

```
public String toString(){
        // code to return a string equivalent of Employee object
}
```

Example 4: Sample code

Change StaffApplication class as:

```
class  StaffApplication
{
        public static void main(String[] args)
        {
                Staff staff = new Staff( );
                System.out.println("Staff details: " + staff);
        }
}
```

Example 5: Sample code

<<TO DO>>

Assignment-5: Complete the following program. **Account.java** contains a partial definition for a class representing a bank account. Place appropriate methods to perform the tasks specified in **ManageAccounts.java**.

For withdrawal check if the balance is sufficient or print appropriate message and give three more chances to the user if the user wishes so.

```
public class Account
{
  private double balance;
  private String name;
  private long accountNumber;

  public Account(double initBal, String name)
  {
    balance = initBal;
  //Initialize name
```



```
    accountNumber =getAccountNnumber();

//account number to be generated randomly and make sure that it is a unique number.
  }

  public String toString()
  {
      // Returns a string containing the name, account number, and balance.
  }

    public void chargeFee()
  {
         // Deducts $5 as bank charges
   }
……
}
public class ManageAccounts
{
    public static void main(String[] args)
    {
      Account account1, account2;
       account1 = new Account(1000, "John", 1111);

      //create account2 of Amith with $500
      //deposit $100 to Amith account
      //print Amith's new balance
      //withdraw $5000 from John's account
      //print John's new balance
      //charge fees to both accounts
      //change the name on Amith's account to John
      //print summary for both accounts
//Test the code exhaustively……
    }
}
```

Example 6: Sample code

Lab 3. Language Fundamentals – Part II

Goals	• At the end of this lab session, you will be able to: ○ Use Arrays ○ Working with array of objects ○ Reuse classes through Inheritance ○ Understand static method and variable and use it ○ Work with strings
Time	2hr 30 min

3.1: Working with Arrays

<<TO DO>>

Assignment-1: Create a class **ArrayDemo**, which behaves like a wrapper around an integer array. The class should have methods to create array, add elements into it, display contents of array, search for an element in the array. Minimum size of the array should be 10.

```
class ArrayDemo{

  int contents []     // declare array.
  ArrayDemo (int){
        // Create array of size <int>
}

  void populateContents(){
        // Populate the array
}

  void showContents(){
        // Display contents of array.
}

 int searchElement(int searchElement){
        // search for element; if found, return its index location, else return –1.
}
}
```

Class UseArray


```
{
        public static void main(String[] args)
        {
                ArrayDemo arrayDemo = new ArrayDemo( );
                // call to the methods in the class.
        }
}
```

Example 7: Sample code

Assignment-2:

Write a java program which will take a list of values from command line and sort them using the following algorithm

- Bubble Sort
- Selection Sort

Example java SortProgram 12 4 15 9 89 90
Should display the output as 4 9 12 15 89 90

Assignment-3:

Write a java code called CurrencyConverter, which will convert Indian Rupees (INR) to

1. GBP
2. US dollars
3. Yen

[Hint: Store all conversion rates into an array]

Assignment-4: Write the program which creates an object of class **Product** that contains details of a product. Create another class called UseProduct to do the following

a**.** Store 20 products

b. Display the product which has the maximum price

```
class Product{
        int productId;
        String description;
        double price;
        int unit;
```



```
Product (){…} //default constructor

Product (int ProdID, String Descr, double Price, int Unit)        //constructor with 3 args

public int getId(){
// return the product id
}

public String getDescription(){
// returns the description of product
}
public double getPrice(){
// returns the price
}

public int getUnit(){
// returns the unit
}
}

Class  UseProduct
{
        public static void main(String[] args)
        {
                Product product  = new Product( );
                //…….


        }}
```

Example 8: Sample code

3.2: Working with enum types

<<TO DO>>

Assignment-5: Create an **enum** type **ArithmeticOperation** which take two numeric arguments as command line arguments, and perform the four basic arithmetic calculations PLUS, MINUS, MULTIPLY, and DIVIDE.

```
public enum ArithmeticOperation {    PLUS, MINUS, TIMES, DIVIDE;       double
eval(double x, double y){   // Do arithmetic op represented by the constants}
```



```
        public static void main(String args[]) {       double x =
Double.parseDouble(args[0]);
       double y = Double.parseDouble(args[1]);
        // Execute the enum for Loop to perform the operations one by one and print out
the results on the console.
   }
}
```

Example 9: Sample code

3.3: Using Static variables and methods.

Solution:

Step 1: Type the following code:

```
public class Demo {
     public Demo() {}

   // Create static variable
   static int staticNumber = 0;

   // Create static method
   static void staticMethod(int formalStaticNumber) {
      System.out.println("staticMethod("+formalStaticNumber +") entered");
   }

   // Create static block
   static {
      System.out.println("Entered :static block , staticNumber = " + staticNumber);
     staticNumber += 1;
      System.out.println("Exiting: static block, staticNumber = " + staticNumber);
   }
} // class ends

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

      // Access a static variable of Demo class.
      System.out.println("Demo. staticNumber = " + Demo. staticNumber);

      // Invoke a static method of Demo class
      Demo.staticMethod(5);

      // The static variable can be accessed from an object instance.
      Demo demo1 = new Demo();
```



```
        System.out.println("demo1.staticNumber = " + demo1. staticNumber);

        // The static method can be invoked from an object instance.
        demo1.staticMethod(0);

        // The same static variable can be accessed from multiple instances.
        Demo demo2 = new Demo();
        System.out.println("demo2.staticNumber = " + demo2. staticNumber);
        demo1.staticNumber += 3;
        System.out.println("Demo. staticNumber = " + Demo. staticNumber);
        System.out.println("demo1.staticNumber = " + demo1. staticNumber);
        System.out.println("demo2. staticNumber = " + demo2. staticNumber);

    }
} //class ends
```

Example 10: StaticDemo.java

Step 2: Execute it. You should get the output as shown below:

```
Entered :static block , staticNumber = 0
Exiting: static block, staticNumber = 1
Demo. staticNumber = 1
staticMethod(5) entered
demo1.staticNumber = 1
staticMethod(0) entered
demo2.staticNumber = 1
Demo. staticNumber = 4
demo1.staticNumber = 4
demo2. staticNumber = 4
```

3.4: Working with Array of Objects

Write a program to create a Salary class.

Solution:

Step 1: Create a class **Salary. Unless mentioned, it is considered that da is 25% of basic and hra is 30% of basic.**

```
public Salary
{

    private int basic;      // members of Salary class

    private double da, hra;         // members of Salary class

        public Salary()
```



```
        {
                // To Do: implement the default constructor
        }



} // end of class Salary ;}
```

Example 11: Salary.java

<<TO DO>>

Assignment-6: Create a class called Employee with the following attributes. Also, add the salary class as a member of the employee class.

```
employeeId
firstName
lastName
address
phoneNumber
dateOfBirth
hireDate
departmentId
```

Example 12:Employee.java

Create the following methods in the Employee class which performs the following

1. Set and Retrieve various data members, the data members should be valid. For example user cannot enter an employee who is below 21
2. Calculate the experience of the employee
3. Calculate the age of the employee
4. Calculating the expected retirement date (dob+58 years) of an employee
5. Find the probation status, an employee is in probation if he has not completed one year in the company.
6. Compare two employees in terms of experience

Create a class called UseEmployee to display the following menu and call the respective methods of Employee class

1. Create new Employees
2. Print Employee details
3. Compare the experience of two Employees
4. Exit

Assignment-7:

To the above employee class, add appropriate constructors .Create an Address class which would have the following data members

- street
- city
- country
- pincode

Replace the address data member in the employee class with the address object and modify the constructors and methods accordingly

3.5: Inheritance

<<TO DO>>

Assignment-8: Look at the Account class and write a main method in a different class to briefly experiment with some instances of the Account class.

```
public class Account
{
	private double balance; //current balance
	private int accNum; //account number
	public Account(int accNum)
{
	this.balance=0.0f;
	this.accNum=accNum;
}
	public void deposit(double sum)
{

		bal-=sum;

}


public double getBalance()
{
	return balance;
}
public double getAccountNumber()
{
	return accNum;
}
	public String toString()
{
	return “Acc”+accNum+”:”+”balance=”+balance;
}
	public final void print()
{
	//Don’t override this,
	//override the toString method
	}
}
```

Example 13:Account.java

Inherit two classes Savings Account and Current Account from the above Account Class. Implement the following in the respective classes.

- **Savings Account**
  a. Add a variable called minimumBalance
  b. Add a method called withdraw (This method should check for minimum balance and allow withdraw to happen)

- **Current Account**
  a. Add a variable called overDraftLimit
  b. Add a method called isOverDraftLimit (checks whether overdraft limit is reached and returns a boolean value accordingly)

Create a new class called Ledger and Implement the following

a. Variables: Array of Accounts, ledgerNumber and ledgerStartDate
b. Methods: printStatement, which will print Account Number, Ledger Number and the balance amount for a given Account Number.

3.6: Working with Strings.

<<TO DO>>

Assignment-9: Write a program to accept a string using the **StringBuilder** class and insert a ***** wherever a vowel appears.

```
class UseStringBuilder{

public static void main(String args[]) {

      // Get input from the user.

       //Create the StringBuilder


      //Append the text entered by the user to the end of the buffer

      // Loop Through to replace vowels using * and print the string . Use the insert and delete methods

      }
}
```

Example 14: Sample code

