**Lesson 1: Introduction to Java**

1. **Responsibilities of Bytecode verifier, class loader, JIT compiler, JVM**

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

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

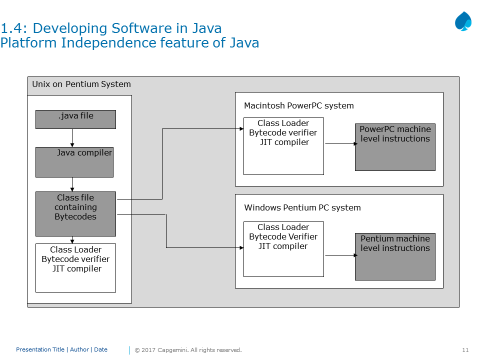
Java’s security model has three primary components:

**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.



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.

***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.*

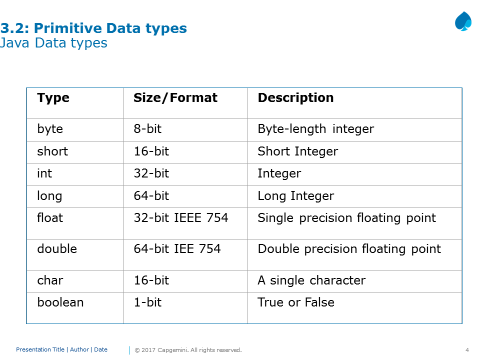
* *Even with the VM, at some point, all Java programs must be resolved to a particular underlying hardware platform. In Java, this resolution occurs within each particular VM implementation. The way this works is that Java programs make calls to the VM, which in turn routes them to appropriate native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

**Lesson 3: Language Fundamentals**

1. **Discuss illegal assignments for e float ,char ,byte data types**

****

There are two data types available in Java:

Primitive Data Types.

Reference/Object Data Types

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

1. **What happens when you ignore break in switch, Discuss the valid data types that w.r.t switch case**

**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.

The switch-expression must evaluate to any of the following type:

* byte
* short
* char
* int
* enum
* String.

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

**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.

**Lesson 4: Classes and Objects**

1. **Discuss all the important points of enum**

Important point to ponder about enums:

* enum can be declared with only a public or default modifier.
* enums are not strings or integer type.
* 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.
* enum cannot be declared in functions.
* A semicolon after an enum is optional

1. **What is the default value of all instance variables**

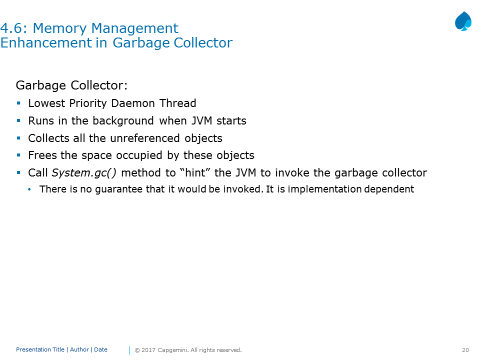
Instance variables have default values. For numbers, the default value is **0**, for Booleans it is false, and for object references it is null. Values can be assigned during the declaration or within the constructor. Instance variables can be accessed directly by calling the variable name inside the class. 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.

**Q) What will happen if you don’t initialize a local variable and try to print it?**

In Java, class and instance variables assume a default value (null, 0, false) if they are not initialized manually. However, local variables don't have a default value. Unless a local variable has been assigned a value, the compiler will refuse to compile the code that reads it.

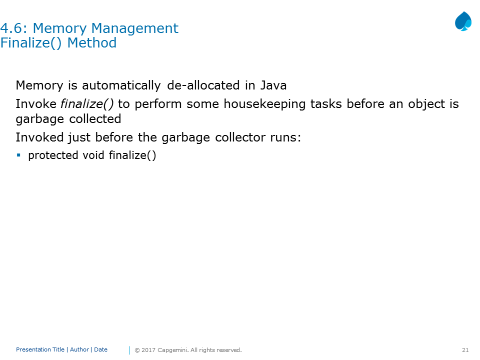
1. **Garbage collector points**
   * **Not assured even if you call System.gc() method ,least priority daemon thread, calls finalize method**

Java Virtual Machine (JVM) de-allocates memory allocated to unreferenced objects during the garbage collection process.



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.

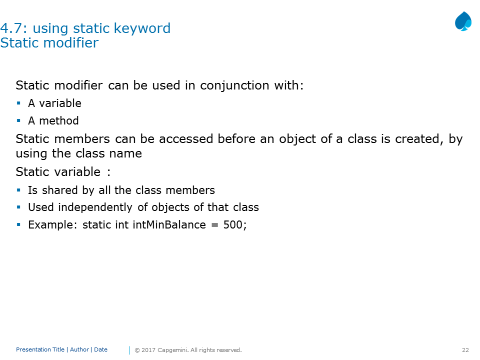


**‘**Protected’ prevents access to finalize() by code defined outside its class.

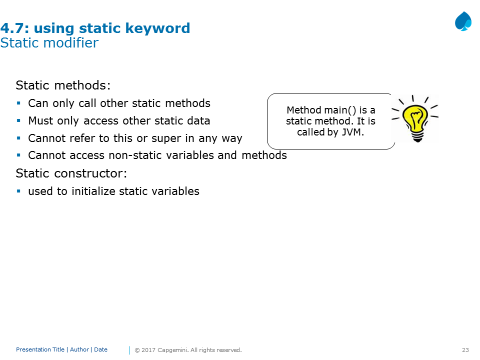
This method only approximates the working of C++’s destructor. There is no way to determine when the finalize() method will run. There is no concept of destructors in Java as is there in C++.

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.

1. **Static methods – discuss all the points**

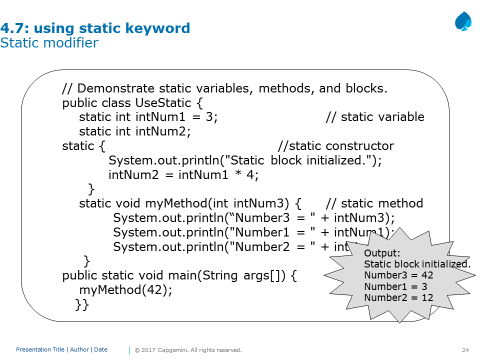


Variables and methods marked with static modifier belong to the class rather than any particular instance. You do not have to instantiate the class to invoke a static method or access a static variable.



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.

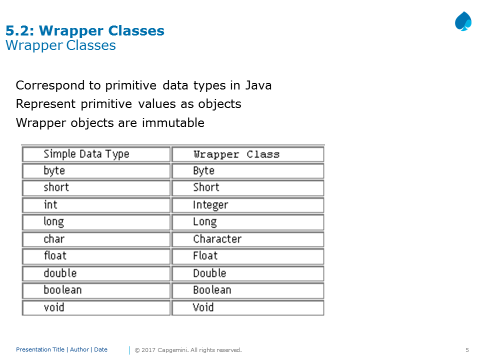
Static constructor is also known as static initialization block. This is a normal block of code enclosed in braces { } and preceded by the static keyword. This can appear anywhere in the class body. It is normally used to initialize static variables.



1. **What is created in heap?**

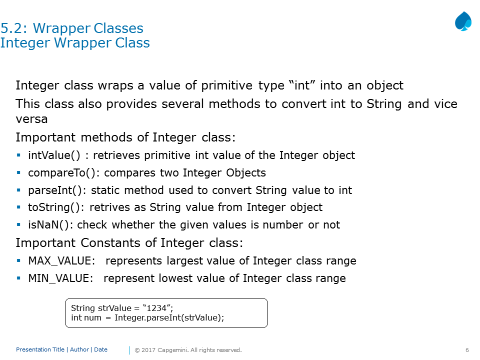
Heap is a memory place where the objects and its instance variable are stored. Also it is to be remembered that the variable references (either primitive or object references) are stored in the stack. (Only for local variables). For instance and class variables, they are part of the object, hence, on the heap.

**Lesson 5: Exploring Basic Java Class Libraries**

1. **What are wrapper classes list all the wrapper classes**

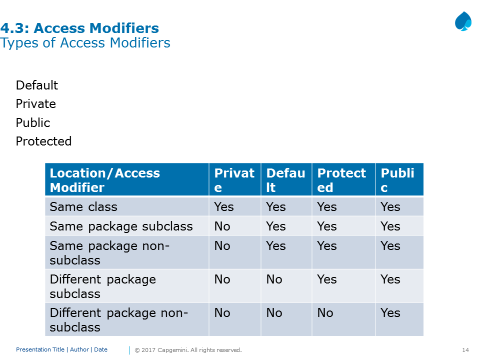
Wrapper classes correspond to the primitive data types in the Java language. These classes represent the primitive values as objects. Wrapper objects are immutable. This means that once a wrapper object has a value assigned to it, that value cannot be changed.

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.



1. **What are the possible modifiers for the top level class, instance variables , local variables**

(Example final is the only modifier used for local variables and public, private and static is used for local)

****

**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**

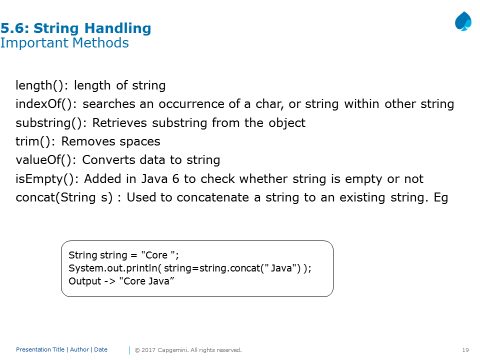
Default specifier is used when “no access modifier is present”. Any class, field, method or constructor that has no declared access modifier is accessible only by classes in the same package. The default modifier is not used for fields and methods within an interface.

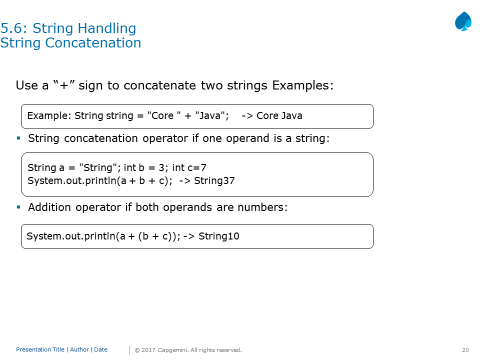
The table shown above is applicable only to members of classes. A class has only two possible access levels: default and public.

When a class is declared as public, it is accessible by any other code.

If a class has default access, then it can only be accessed by other code within its same package

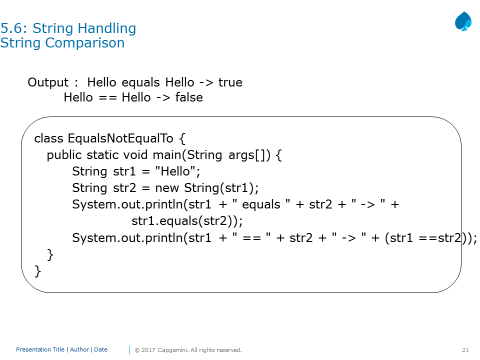
1. **Which type of variables must be initialized-mandatory(ans: final variable)**
2. **String ,string buffer and string builder – discuss which is mutable and methods ( append, concat , etc), equlas and == w.r.t String ?**

****

****

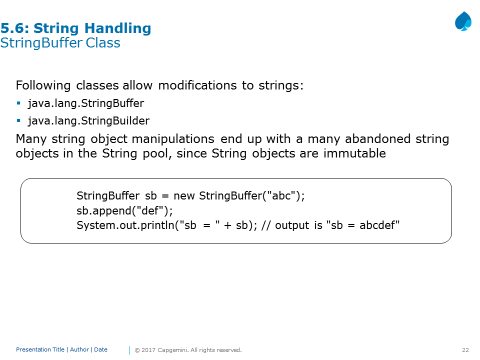
The concat() method seen in previous page allows one string to be concatenated to another. But Java also supports string concatenation with the “+” operator.

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.

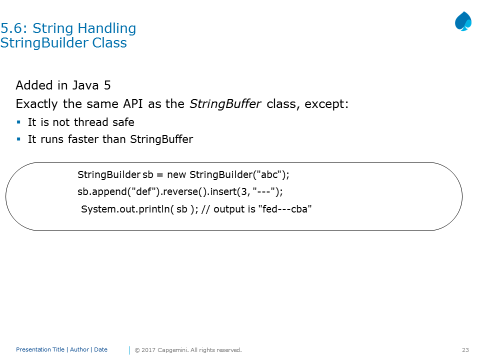


The String class includes various methods that compare strings or substrings within each string. The most popularly used two ways to compare the strings is either using = = operator or by using the equals method.

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.



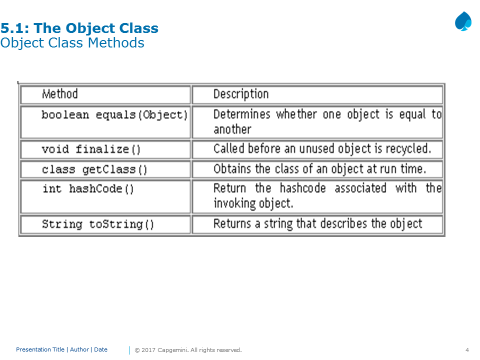
We got a nice new String out of the deal, but the downside is that the old String "abc" has been lost in the String pool, thus wasting memory. If we were using a String Buffer instead of a String, the code would look like this.



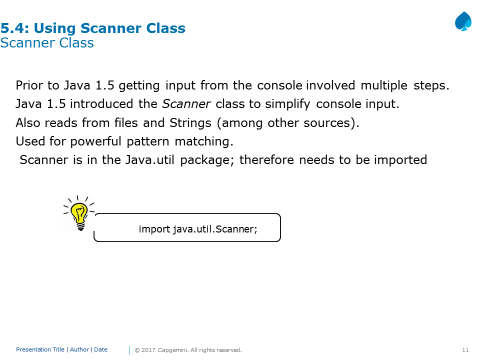
The StringBuilder class was added in Java 5. It has exactly the same API as the StringBuffer class, except StringBuilder is not thread safe. In other words, its methods are not synchronized. Sun recommends that you use StringBuilder instead of StringBuffer whenever possible because StringBuilder will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

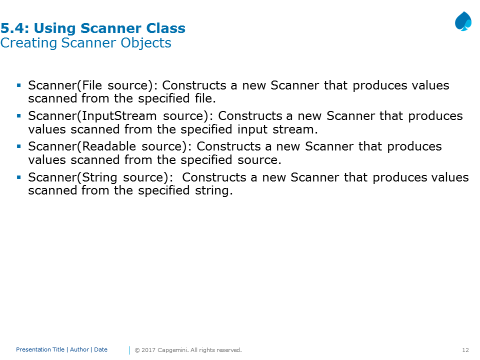
Synchronization, anything we say about StringBuilder's methods holds true for StringBuffer's methods, and vice versa.

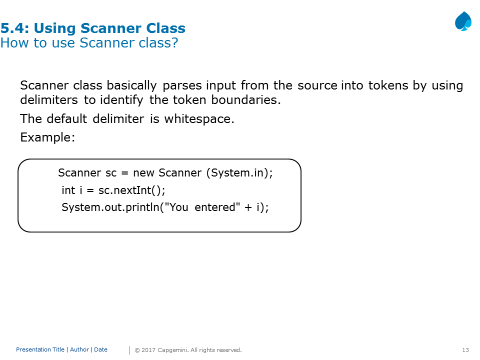
1. **Methods of object class- list out**

****

1. **Scanner ,use delimiter method**

****

****

****

**Tokens:**

String next(): Finds and returns the next complete token from this scanner.

boolean nextBoolean(): Scans the next token of the input into a boolean value and returns that value.

byte nextByte(): Scans the next token of the input as a byte.

double nextDouble(): Scans the next token of the input as a double.

float nextFloat(): Scans the next token of the input as a float.

int nextInt(): Scans the next token of the input as an int.

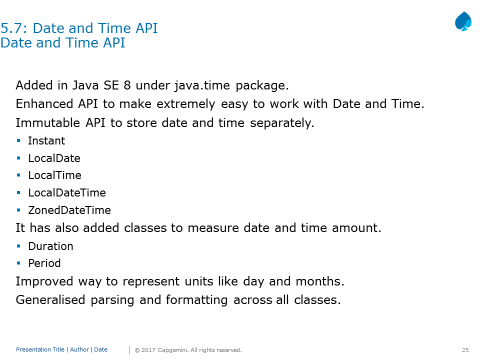
String nextLine(): Advances this scanner past the current line and returns the input that was skipped.

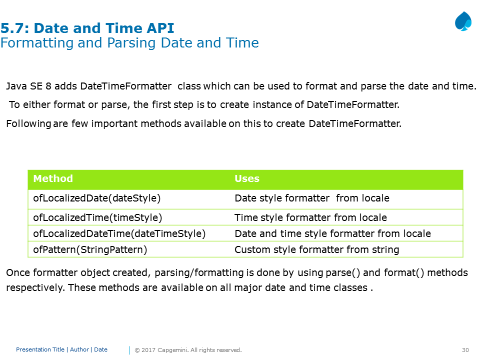
long nextLong(): Scans the next token of the input as a long.

short nextShort(): Scans the next token of the input as a short.

InputMismatchException: This exception can be thrown if you try to get the next token using a next method that does not match the type of the token.

1. **All new date features, LocalDate methods to get current date, tomorrows date, yesterdays**

****

****

1. **Discuss equals and hashCode()**

In hashCode() it says: If two objects are equal according to the equals(Object) method, then calling the hashCode method on each of the two objects must produce the same integer result. If you only override equals() and not hashCode()your class violates this contract.