**Java Applets**: An applet is a special kind of Java program that is designed to be transmitted over the Internet and automatically executed by a Java-compatible web browser.

**Bytecode-** Bytecode is a highly optimized set of instructions designed to be executed by the Java run-time system, which is called the Java Virtual Machine (JVM). In essence, the original JVM was designed as an interpreter for bytecode

**Buzzwords:**

• **Simple** - Java was designed to be easy for the professional programmer to learn and use effectively

**• Secure -** Java achieved this protection by confining an applet to the Java execution environment and

not allowing it access to other parts of the computer.

**• Portable -** The same code must work on all computers. Therefore, some means of generating portable

executable code was needed.

**• Object-oriented-** The object model in Java is simple and easy to extend.

**• Robust -** The multiplatformed environment of the Web places extraordinary demands on a program,

because the program must execute reliably in a variety of systems. Thus, the ability to create

robust programs was given a high priority in the design of Java.

**• Multithreaded-** Java was designed to meet the real-world requirement of creating interactive,

networked programs. To accomplish this, Java supports multithreaded programming,

which allows you to write programs that do many things simultaneously.

**• Architecture-neutral -** “write once; run anywhere, any time, forever.”

**• Interpreted and High performance-** Java enables the creation of cross-platform programs by .

compiling into an intermediate representation called Java bytecode. This code can be

executed on any system that implements the Java Virtual Machine.

**• Distributed-** Java is designed for the distributed environment of the Internet because it handles TCP/IP

protocols.

**• 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.

The keyword static allows main( ) to be called without having to instantiate a particular instance of the class. This is necessary since main( ) is called by the Java Virtual Machine before any objects are made.

Objects of type String store character strings. In this case, args receives any command-line arguments present when the program is executed.

Java is a free-form language. This means that you do not need to follow any special indentation rules.

**Java Keywords**

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

**The Primitive Types**

Java defines eight primitive types of data: byte, short, int, long, char, float, double, and boolean.

**The For-Each Version of the for Loop**

for(int i=0; i < 10; i++) sum += nums[i];

for(int x: nums) sum += x;

**Declaring Objects**

Box mybox = new Box();

This statement combines the two steps just described. It can be rewritten like this to show each step more clearly:

Box mybox; // declare reference to object

mybox = new Box(); // allocate a Box object

**The this Keyword**

this can be used inside any method to refer to the current object. That is, this is always a reference to the object on which the method was invoked. You can use this anywhere a reference to an object of the current class’ type is permitted.

**Garbage Collection**

Since objects are dynamically allocated by using the new operator, you might be wondering how such objects are destroyed and their memory released for later reallocation. In some languages, such as C++, dynamically allocated objects must be manually released by use of a delete operator. Java takes a different approach; it handles deallocation for you automatically. The technique that accomplishes this is called garbage collection. It works like this: when no references to an object exist, that object is assumed to be no longer needed, and the memory occupied by the object can be reclaimed. There is no explicit need to destroy objects as in C++. Garbage collection only occurs sporadically (if at all) during the execution of your program.

**Introducing final**

A variable can be declared as final. Doing so prevents its contents from being modified. This means that you must initialize a final variable when it is declared. For example:

final int FILE\_NEW = 1;

final int FILE\_OPEN = 2;

**The finalize( ) Method**

Sometimes an object will need to perform some action when it is destroyed. For example, if an object is holding some non-Java resource such as a file handle or character font, then you might want to make sure these resources are freed before an object is destroyed. To handle 122 Part I: The Java Language such situations, Java provides a mechanism called finalization. By using finalization, you can define specific actions that will occur when an object is just about to be reclaimed by the garbage collector.

**Overloading Methods**

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. When this is the case, the methods are said to be overloaded, and the process is referred to as method overloading.

**Method Overriding**

In a class hierarchy, when a method in a subclass has the same name and type signature as a method in its super class, then the method in the subclass is said to override the method in the super class.

Method overriding occurs only when the names and the type signatures of the two methods are identical. If they are not, then the two methods are simply overloaded.

Overridden methods allow Java to support run-time polymorphism.

**Using final to Prevent Overriding**

While method overriding is one of Java’s most powerful features, there will be times when you will want to prevent it from occurring. To disallow a method from being overridden, specify final as a modifier at the start of its declaration. Methods declared as final cannot be overridden.

**Static**

The most common example of a static member is main( ). main( ) is declared as static because it must be called before any objects exist.

**Using Command-Line Arguments**

Sometimes you will want to pass information into a program when you run it. This is accomplished by passing command-line arguments to main( ). A command-line argument is the information that directly follows the program’s name on the command line when it is executed. To access the command-line arguments inside a Java program is quite easy— they are stored as strings in a String array passed to the args parameter of main( ). The first command-line argument is stored at args[0], the second at args[1], and so on.

**Super**

Whenever a subclass needs to refer to its immediate superclass, it can do so by use of the keyword super. Super has two general forms. The first calls the superclass’ constructor. The second is used to access a member of the superclass that has been hidden by a member of a subclass.

**Abstract**

Any class that contains one or more abstract methods must also be declared abstract. To declare a class abstract, you simply use the abstract keyword in front of the class keyword at the beginning of the class declaration. There can be no objects of an abstract class. That is, an abstract class cannot be directly instantiated with the new operator. Such objects would be useless, because an abstract class is not fully defined. Also, you cannot declare abstract constructors, or abstract static methods. Any subclass of an abstract class must either implement all of the abstract methods in the superclass, or be itself declared abstract.

**Dynamic method dispatch**

Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time. Dynamic method dispatch is important because this is how Java implements run-time polymorphism

**String buffer**

Stringbuffer is a peer class of String that provides much of the functionality of strings. As you know, String represents fixed-length, immutable character sequences. In contrast, stringbuffer represents growable and writeable character sequences. Stringbuffer may have characters and substrings inserted in the middle or appended to the end. Stringbuffer will automatically grow to make room for such additions and often has more characters preallocated than are actually needed, to allow room for growth.

**String Builder**

J2SE 5 adds a new string class to Java’s already powerful string handling capabilities. This new class is called String Builder. It is identical to String Buffer except for one important difference: it is not synchronized, which means that it is not thread-safe. The advantage of String Builder is faster performance. However, in cases in which you are using multithreading, you must use String Buffer rather than StringBuilder.S

**JAVA PACKAGES**

A java package is a group of similar types of classes, interfaces and sub-packages. Package in java can be categorized in two form, built-in package and user-defined package. There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

package mypack;

public class Simple{

  public static void main(String args[]){

    System.out.println("Welcome to package");

 }

}

How to compile java package

javac -d directory javafilename

javac -d . Simple.java

NOTE:- The -d switch specifies the destination where to put the generated class file. You can use any directory name like /home (in case of Linux), d:/abc (in case of windows) etc. If you want to keep the package within the same directory, you can use . (dot)

## How to run java package program

|  |
| --- |
| **To Compile:** javac -d . Simple.java |
| **To Run:** java mypack.Simple (java packagename.javafilename) How to access package from another package?1) Using packagename.\*  1. //save by A.java 2. package pack; 3. public class A{ 4. public void msg(){System.out.println("Hello");} 5. } 6. //save by B.java 7. package mypack; 8. import pack.\*; 10. class B{ 11. public static void main(String args[]){ 12. A obj = new A(); 13. obj.msg(); 14. } 15. }  2) Using packagename.classname  1. //save by A.java 3. package pack; 4. public class A{ 5. public void msg(){System.out.println("Hello");} 6. } 7. //save by B.java 8. package mypack; 9. import pack.A; 11. class B{ 12. public static void main(String args[]){ 13. A obj = new A(); 14. obj.msg(); 15. } 16. }  3) Using fully qualified name  1. //save by A.java 2. package pack; 3. public class A{ 4. public void msg(){System.out.println("Hello");} 5. } 6. //save by B.java 7. package mypack; 8. class B{ 9. public static void main(String args[]){ 10. pack.A obj = new pack.A();//using fully qualified name 11. obj.msg(); 12. } 13. }   **INTERFACES**  An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.  An interface is similar to a class in the following ways −   * An interface can contain any number of methods. * An interface is written in a file with a **.java** extension, with the name of the interface matching the name of the file. * The byte code of an interface appears in a **.class** file. * Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.   However, an interface is different from a class in several ways, including −   * You cannot instantiate an interface. * An interface does not contain any constructors. * All of the methods in an interface are abstract. * An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final. * An interface is not extended by a class; it is implemented by a class. * An interface can extend multiple interfaces.  1. interface printable{ 2. void print(); 3. } 4. class A6 implements printable{ 5. public void print(){System.out.println("Hello");} 7. public static void main(String args[]){ 8. A6 obj = new A6(); 9. obj.print(); 10. } 11. }  Default Method in Interface  1. interface Drawable{ 2. void draw(); 3. default void msg(){System.out.println("default method");} 4. } 5. class Rectangle implements Drawable{ 6. public void draw(){System.out.println("drawing rectangle");} 7. } 8. class TestInterfaceDefault{ 9. public static void main(String args[]){ 10. Drawable d=new Rectangle(); 11. d.draw(); 12. d.msg(); 13. }}  Static Method in Interface  1. interface Drawable{ 2. void draw(); 3. static int cube(int x){return x\*x\*x;} 4. } 5. class Rectangle implements Drawable{ 6. public void draw(){System.out.println("drawing rectangle");} 7. } 9. class TestInterfaceStatic{ 10. public static void main(String args[]){ 11. Drawable d=new Rectangle(); 12. d.draw(); 13. System.out.println(Drawable.cube(3)); 14. }}  Interfaces Can Be ExtendedOne interface can inherit another by use of the keyword extends. The syntax is the same as for inheriting classes. When a class implements an interface that inherits another interface, it must provide implementations for all methods defined within the interface inheritance chain. Following is an example:// One interface can extend another.interface A { void meth1();void meth2(); }// B now includes meth1() and meth2() -- it adds meth3(). interface B extends A {void meth3(); }// This class must implement all of A and B class MyClass implements B The Throws/Throw Keywords  If a method does not handle a checked exception, the method must declare it using the **throws** keyword. The throws keyword appears at the end of a method's signature.  You can throw an exception, either a newly instantiated one or an exception that you just caught, by using the **throw** keyword.  Try to understand the difference between throws and throw keywords, *throws* is used to postpone the handling of a checked exception and *throw* is used to invoke an exception explicitly.  The Finally Block  The finally block follows a try block or a catch block. A finally block of code always executes, irrespective of occurrence of an Exception.  Using a finally block allows you to run any cleanup-type statements that you want to execute, no matter what happens in the protected code.  A finally block appears at the end of the catch blocks THROW  1. **public** **class** TestThrow1{ 2. **static** **void** validate(**int** age){ 3. **if**(age<18) 4. **throw** **new** ArithmeticException("not valid"); 5. **else** 6. System.out.println("welcome to vote"); 7. } 8. **public** **static** **void** main(String args[]){ 9. validate(13); 10. System.out.println("rest of the code..."); 11. } 12. } | |