# Term table with term and score

|  |  |  |  |
| --- | --- | --- | --- |
| Term | Subject id | Frequency |  |
| program | 4 | 4 |
| will | 4 | 4 |
| core | 4 | 4 |
| need | 4 | 3 |
| be | 4 | 4 |
| code | 4 | 4 |
| examples | 4 | 3 |
| have | 4 | 5 |
| programming | 4 | 4 |
| start | 4 | 3 |
| is | 4 | 5 |
| how | 4 | 3 |
| basic | 4 | 3 |
| creating | 4 | 3 |
| are | 4 | 5 |
| platform | 4 | 3 |
| time | 4 | 3 |
| other | 4 | 4 |
| language | 4 | 5 |
| was | 4 | 3 |
| programs | 4 | 3 |
| run | 4 | 4 |
| has | 4 | 5 |
| applications. | 4 | 3 |
| use | 4 | 4 |
| dynamic | 4 | 3 |
| there | 4 | 3 |
| computer | 4 | 3 |
| mainly | 4 | 3 |
| called | 4 | 5 |
| source | 4 | 4 |
| high | 4 | 3 |
| machine | 4 | 4 |
| written | 4 | 3 |
| compiled | 4 | 3 |
| compiler | 4 | 4 |
| runs | 4 | 3 |
| can | 4 | 4 |
| system | 4 | 3 |
| first | 4 | 4 |
| known | 4 | 3 |
| software | 4 | 4 |
| tools | 4 | 3 |
| do | 4 | 3 |
| install | 4 | 3 |
| file | 4 | 4 |
| name | 4 | 4 |
| execute | 4 | 3 |
| environment | 4 | 4 |
| using | 4 | 3 |
| also | 4 | 4 |
| used | 4 | 4 |
| create | 4 | 3 |
| write | 4 | 4 |
| program. | 4 | 3 |
| public | 4 | 5 |
| class | 4 | 5 |
| { | 4 | 3 |
| static | 4 | 4 |
| void | 4 | 4 |
| } | 4 | 4 |
| just | 4 | 3 |
| open | 4 | 3 |
| java | 4 | 5 |
| not | 4 | 3 |
| error | 4 | 3 |
| concept | 4 | 3 |
| object | 4 | 3 |
| objects | 4 | 3 |
| types | 4 | 3 |
| general | 4 | 3 |
| later | 4 | 3 |
| now | 4 | 3 |
| new | 4 | 3 |
| created | 4 | 3 |
| let's | 4 | 3 |
| variable | 4 | 3 |
| next | 4 | 4 |
| an | 4 | 3 |
| in | 4 | 4 |
| over | 4 | 3 |
| the | 4 | 4 |
| which | 4 | 4 |
| to | 4 | 4 |
| a | 4 | 4 |
| java, | 4 | 3 |
| of | 4 | 4 |
| that | 4 | 4 |
| this | 4 | 3 |
| any | 4 | 3 |
| for | 4 | 4 |
| and | 4 | 4 |
| we | 4 | 3 |
| it | 4 | 3 |
| with | 4 | 3 |
| on | 4 | 4 |
| since | 4 | 3 |
| you | 4 | 3 |
| history | 4 | 3 |
| from | 4 | 3 |
| java. | 4 | 3 |
| as | 4 | 3 |
| runtime | 4 | 3 |
|  |  |  |  |
|  |  |  |  |

# Data Table holding relevant pages



# Term Data Table holding pages with terms

|  |  |  |
| --- | --- | --- |
| Data id | S id | Data |
| 11 | 4 | So, you want to program in Java? That's great, and you've come to the right place. This column will give you a complete education on Java programming, starting with the basics, and covering all of the core concepts you need to become productive in the language. This column will be technical, with plenty of code examples to help you along. I'll assume that you already have some programming experience. I'll start off with a short overview of what Java is and how it works, move on to basic coverage of some object-oriented programming concepts, and then jump right into creating Java classes -- the heart of Java programming. I'll try to give examples and instructions that are as platform-neutral as possible, but I'll default to the Windows platform when necessary. Unix users should have an easy time interpreting these examples for the Unix world. Mac and other users will have to work a little harder (our apologies). High-level overview of Java Java is a general-purpose, object-oriented language that looks a lot like C and C++. Its design, however, was built on those of its predecessors, making it easier, safer, and more productive than C++. While Java started out as a niche language for developing applets or small programs that run in Web browsers, it has evolved to arguably become the most important programming language for developing ecommerce and other Web-driven applications. Its area of use is growing daily and includes dynamic Web-content generation with servlet technology, the building of business components with Enterprise JavaBeans, the creation of cross-platform user interfaces with Swing, and much more. Portable, distributed, multitier, object-oriented programs driven by the Web are the order of the day, and there is no language better than Java for writing these programs. The Java Virtual Machine Let's take a look at a central component of the Java architecture, the Java Virtual Machine (JVM). The JVM is what gives Java its cross-platform functionality and many of its security and safety capabilities. The JVM is basically an abstract computer implemented in software. I'll focus mainly on its instruction set, which is called bytecode. Bytecode is an intermediate language between Java source and the target computer you want to run on. The following figure demonstrates how it works at a very high level. From Java source to bytecode to host machine code Programs are written in Java and stored in .java files (for example, MyClass.java) The .java files are compiled by the Java compiler into bytecode and stored in .class files (for example, MyClass.class) The JVM loads the bytecode (the .class files), performs some checks on it, and then converts it to the machine code of the target platform that executes it This is where Java gets its platform independence. The bytecode format is the same on all platforms because it runs in the same abstract machine -- the JVM. As long as there is a JVM on any given platform, you can run Java on it. There's an old saying in computer programming, "You can solve any problem with another level of indirection." The JVM and bytecode together is another level of indirection. That's about it on the JVM for now. If you want to know more, check out Sun's JVM specification in the Resources section below. Setting up Java on your system The first thing you need is the Java 2 Software Development Kit or SDK (formerly known as the Java Development Kit, or JDK). This is a set of software and software tools supplied by Sun that includes all of the basic components needed to build Java programs. If you don't have this, you'll need to download the Java 2 SDK from Sun (see Resources below). Here's a brief description of what you need to do to install version 1.2.x for the Windows platform: For Windows, the download will be a self-extracting archive, and the file name follows the form: jdk<version>-win.exe. For example, for version 1.2.2, the file is named jdk1\_2\_2-win.exe. Execute the program to install the Java 2 SDK. You can execute it by typing its name at a command prompt. You can double-click it from Explorer. On Windows, the default installation directory is C:\JDK1.2.x You must set up your environment correctly Your path must be set to include the bin subdirectory For example, if you installed the Java 2 SDK in C:\JDK1.2.x, you must include C:\JDK1.2.x\bin in your path If you are using Windows 95/98, you can do this in autoexec.bat If you are using Windows NT, go to Control Panel, System, Environment You can set environment variables in the dialog box there There are other parts of the environment that are important, especially the CLASSPATH environment variable. I'll discuss this in future columns. It's also important to download the javadoc HTML documentation for the core API classes. This is a separate download, in zip format, and the filename is in the form of jdk<version>-doc.zip (for example, jdk1\_2\_2-doc.zip). The javadoc HTML documentation should be unzipped into the Java 2 SDK installation directory Note that the zip file has a directory structure within it, so that unzipping it into C:\ will place all of the files at the root directory C:\JDK1.2.x\docs, which is a good place for them. The following tools will be used to create and run our first program: javac: the Java compiler that converts Java source to bytecode java: the Java Virtual Machine How to write a Java program OK, it's time to write our first Java program. A Java program looks like this: public class MyProgram { public static void main(String[] args) { System.out.println( "Eureka, I can put Java on my resume."); } } A Java program is a class with a main method in it. The main method is a special method that is the starting point for every Java application. It has to be declared exactly as above, and it has to appear within a class, as the above example also shows. System.out.println is the magic incantation you use to get things sent to the console. This code example is for a standalone, nongraphical Java program that prints the string Eureka, I can put Java on my resume. to the console. Your salary can double now, so if you want, you can stop here. Those that want to learn more can keep going. Type the above program into a file named MyProgram.java. It's a good idea to put it in its own directory just to keep things organized. Use your favorite text editor for this, or check out the Resources section for some of my favorites. Once you've done that, go to a command line that is open to the location your file is in. Type in the following code to compile the file. (This generates the .class file from the .java file): javac MyProgram.java Once it compiles without errors, do a directory listing to see that you've generated a MyProgram.class file. (Hey -- that's the bytecode.) Now type the following to run the program: java MyProgram This executes the JVM, and makes it run the bytecode in the MyProgram.class file. You should see the following output on your screen: Eureka, I can put Java on my resume. Congratulations! You've generated your first Java program. There are a couple of things you should notice. The Java compiler (javac) requires that you include the .java extension for the files you're compiling. It's just a program, and that's what it expects you to give it. The JVM (java) does not expect you to include the .class extension. Try typing java MyProgram.class, and you'll get an error message because it will look for a file called MyProgram.class.class. Note also that things are set up in such a way that you have to do everything from one directory. Object-oriented programming (OOP) Object-oriented programming is the key organizational factor in using Java well. Simply put, this means that you organize your program around objects. What's an object? It's a representation of some concept or thing. This is a powerful model because it reflects the way we think about the world, and the way we conceptualize it using language. Here are the key characteristics that define an object: Behavior: What an object can do, or what can be done to it Properties: Data or information associated with an object (often called state) Type: The kind (or class) of thing an object is Identity: The unique existence of every object, independent of its characteristics You might see object described elsewhere using different terminology, but don't worry too much about that. Grasping the concepts is the important thing. Let's further explore the notion of type. We're almost always dealing with many different objects, and the natural inclination is to classify or label them in some way. A type is the way to classify objects, and it's also the blueprint that defines the behavior and properties for objects of a certain type. Here are some real-world examples of different types from different areas: Concrete: Person, car, alarm clock, planet, star Conceptual: Number, democracy Events: General protection fault, earthquake Every object has a type associated with it, and possibly more than one type. Creating types and classifying your objects is one of the arts of OOP, and there is no single "correct" way to do this. Let's examine the alarm clock type. We'll take a crack at determining its behavior and properties. Alarm clock properties: Current time, alarm time Alarm clock behavior: Ring, snooze, turn alarm on or off Alarm clock property-related behavior: Get and set current time; get and set alarm time We've made a bit of a distinction here between general behavior, and behavior that is just related to getting and setting properties. We'll talk more about this later when we delve into the best way to actually program a type. How do we use this type? Easy. We buy an alarm clock and bring it home. We set the current time and the alarm time. When it rings, we hit the snooze button (repeat five to six times), and we finally turn the alarm off and get out of bed. The system depicted in this example is the core of OOP. We have types that represent the different kinds of things in our system, (the alarm clock type), and we have objects that are instances or examples of the types (we bought a particular alarm clock). Objects interact through their behaviors, sometimes resulting in changes in their properties, and ultimately (hopefully) do useful work. C'mon, you already knew this, otherwise you wouldn't be able to use your alarm clock, and you'd never get to work on time. Rewrite your resume -- you're an object-oriented programmer. OOP and Java We've been talking abstractly until now about OOP. Let's shift back and see how OOP is done in Java. A class is used to define a new type in Java, and classes are central to the language. Everything happens within classes. All executable code exists in classes (generally in methods), and all data appears within a class (in variables within a class, and in variables in methods). Classes are defined in a class definition. Here's a (simple and incomplete) definition for an AlarmClock class. public class AlarmClock { } The class keyword introduces the class AlarmClock is the name of the class { starts the class body The data and methods are declared within the class body } ends the class definition We'll look at what public means shortly It's time to create your first class, albeit one that doesn't do much yet. It will soon, though. Create a file called AlarmClock.java, and put in the code for the class definition above. Compile it using javac. javac AlarmClock.java Note that when a class definition starts with the keyword public, Java requires you to put the class definition (the Java source) into a file that starts with the name of the class. Hence, we name our file AlarmClock.java because the class name is AlarmClock. What can we do with this? Have we actually created an AlarmClock? Nope. Remember that a class (that defines a type in Java) is a blueprint for an object, but it's not the actual object. A class definition is like a cookie cutter, but it's not a cookie. You create cookies by cutting some dough with the cookie cutter. Similarly, we have to cut some objects from memory to create them. You create or instantiate objects in Java with an operator called new followed by the name of the class for which you want to create an instance. The syntax looks like this: new AlarmClock(); This creates an alarm clock off the heap. There's still a problem here, however. Unlike cookies we can just pick up and put on a cookie sheet (and then pick up and eat), objects float around in a computer's memory, and we need to have a way to get to them. Java uses special variables called object references to refer to objects. You can initialize a reference to refer to a given object, and then use the reference to interact with that object. Here's what this looks like: AlarmClock aClock = new AlarmClock(); The first AlarmClock declares the type of the variable; aClock is the name of the variable; and new AlarmClock() creates a new alarm clock that is then used to initialize aClock. Just for fun, let's add a little behavior to our alarm clock. You add behavior in methods. We'll add the snooze method, which will just print a message to the console as our first program did. Methods go inside classes, and here's what one looks like: public void snooze() { System.out.println("ZZZZZ"); } When we put it in our AlarmClock class, we get the following: public class AlarmClock { public void snooze() { System.out.println("ZZZZZ"); } } Does AlarmClock.java contain a program? No, it doesn't, because there is no main method in class AlarmClock. Remember that a Java program is just a class with a main method in it. Let's create another program called AlarmClockTest. This just means we'll create a class AlarmClockTest with a main method in it. We'll create an instance of AlarmClock in this program. public class AlarmClockTest { public static void main(String[] args) { AlarmClock aClock = new AlarmClock(); } } Last, but not least, we want to do work with this alarm clock. The way you tell an object to do something is by calling, or, invoking, one of its methods. You generally do this through the variable that refers to it, and you use the dot operator for this. All we can do is snooze with this clock, but hey, that's everybody's favorite action with an alarm clock anyway. Here's what the code looks like: public class AlarmClockTest { public static void main(String[] args) { AlarmClock aClock = new AlarmClock(); aClock.snooze(); } } Ready? Create AlarmClockTest.java, and type in the code above. Compile it (javac AlarmClockTest.java) as well as AlarmClock.java. You're ready to run your new program. The main method is in AlarmClockTest class, so this is how you run the program: java AlarmClockTest You should see ZZZZZ printed out on the command line. Congratulations, you've now completed your second Java program! Review We covered quite a bit here. We looked at the roots of Java, and some of its underlying architecture. We've examined the basics of object-oriented programming and how you use OOP in Java. We also created a Java class, and we've written a couple of short Java programs. We skipped over a lot of detail and moved very quickly so we could get to some of the more interesting aspects of Java programming, like creating classes and calling methods, in this first column. I'll be going into much more detail on all of this in the future. In my next column, I'll look at more OOP concepts, such as encapsulation and abstraction, and expand on how you can use data/variables in Java. We'll add more functionality to our AlarmClock class, and we'll begin to look at how to build classes. For homework, you can take a look at the documentation that comes with Java. You did install that on your system, right? If you installed Java in C:\jdk1.2.2, then the API docs are in file:///C:/Apps/java/docs/api/index.html. Go there and look for the String and Calendar classes in the frame titled "All Classes." We'll be using both of these classes soon, and it's a good idea to start getting used to this documentation. It will be your friend. Enjoy! Jacob Weintraub is founder and president of LearningPatterns.com (LPc). Jacob has been working in object technologies since 1989, and teaching Java since 1995. He authored LPc's Java for Programmers, as well as many of its advanced courses, such as those on OOAD and EJB. |
| 12 | 4 | Hello, World! Java is an object oriented language (OOP). Objects in Java are called "classes". Let's go over the Hello world program, which simply prints "Hello, World!" to the screen. public class Main { public static void main(String[] args) { System.out.println("Hello, World!"); } } Execute Code The first line defines a class called Main. public class Main { Execute Code In Java, every line of code that can actually run needs to be inside a class. This line declares a class named Main, which is public, that means that any other class can access it. This is not important for now, so don't worry. For now, we'll just write our code in a class called Main, and talk about objects later on. Notice that when we declare a public class, we must declare it inside a file with the same name (Main.java), otherwise we'll get an error when compiling. When running the examples on the site, we will not use the public keyword, since we write all our code in one file. The next line is: public static void main(String[] args) { Execute Code This is the entry point of our Java program. the main method has to have this exact signature in order to be able to run our program. public again means that anyone can access it. static means that you can run this method without creating an instance of Main. void means that this method doesn't return any value. main is the name of the method. The arguments we get inside the method are the arguments that we will get when running the program with parameters. It's an array of strings. We will use it in our next lesson, so don't worry if you don't understand it all now. System.out.println("Hello, World!"); Execute Code System is a pre-defined class that Java provides us and it holds some useful methods and variables. out is a static variable within System that represents the output of your program (stdout). println is a method of out that can be used to print a line. Exercise Print "Hello, World!" to the console. |
| 13 | 4 | Introduction to Java 1.1. History Java is a programming language created by James Gosling from Sun Microsystems (Sun) in 1991. The first publicly available version of Java (Java 1.0) was released in 1995. Sun Microsystems was acquired by the Oracle Corporation in 2010. Oracle has now the steermanship for Java. Over time new enhanced versions of Java have been released. The current version of Java is Java 1.7 which is also known as Java 7. From the Java programming language the Java platform evolved. The Java platform allows software developers to write program code in other languages than the Java programming language which still runs on the Java virtual machine. The Java platform is usually associated with the Java virtual machine and the Java core libraries. 1.2. Java and Open Source In 2006 Sun started to make Java available under the GNU General Public License (GPL). Oracle continues this project called OpenJDK. 1.3. Java virtual machine The Java virtual machine (JVM) is a software implementation of a computer that executes programs like a real machine. The Java virtual machine is written specifically for a specific operating system, e.g., for Linux a special implementation is required as well as for Windows. Java programs are compiled by the Java compiler into bytecode. The Java virtual machine interprets this bytecode and executes the Java program. 1.4. Java Runtime Environment vs. Java Development Kit A Java distribution typically comes in two flavors, the Java Runtime Environment (JRE) and the Java Development Kit (JDK). The Java runtime environment (JRE) consists of the JVM and the Java class libraries. Those contain the necessary functionality to start Java programs. The JDK additionally contains the development tools necessary to create Java programs. The JDK therefore consists of a Java compiler, the Java virtual machine and the Java class libraries. |
| 15 | 4 | Java Tutorial Java - What, Where and Why? What is Java Where Java is used Java Applications Java Tutorial or Core Java Tutorial or Java Programming Tutorial is a widely used robust technology. Let's start learning of java from basic questions like what is java tutorial, core java, where it is used, what type of applications are created in java and why use java. What is Java Java is a programming language and a platform. Java is a high level, robust, secured and object-oriented programming language. Platform: Any hardware or software environment in which a program runs, is known as a platform. Since Java has its own runtime environment (JRE) and API, it is called platform. Java Example Let's have a quick look at java programming example. A detailed description of hello java example is given in next page. class Simple{ public static void main(String args[]){ System.out.println("Hello Java"); } } Test it Now Where it is used? According to Sun, 3 billion devices run java. There are many devices where java is currently used. Some of them are as follows: Desktop Applications such as acrobat reader, media player, antivirus etc. Web Applications such as irctc.co.in, javatpoint.com etc. Enterprise Applications such as banking applications. Mobile Embedded System Smart Card Robotics Games etc. Types of Java Applications There are mainly 4 type of applications that can be created using java programming: 1) Standalone Application It is also known as desktop application or window-based application. An application that we need to install on every machine such as media player, antivirus etc. AWT and Swing are used in java for creating standalone applications. 2) Web Application An application that runs on the server side and creates dynamic page, is called web application. Currently, servlet, jsp, struts, jsf etc. technologies are used for creating web applications in java. 3) Enterprise Application An application that is distributed in nature, such as banking applications etc. It has the advantage of high level security, load balancing and clustering. In java, EJB is used for creating enterprise applications. 4) Mobile Application An application that is created for mobile devices. Currently Android and Java ME are used for creating mobile applications. Do You Know ? What is the difference between JRE and JVM ? What is the purpose of JIT compiler ? Can we save the java source file without any name ? Why java uses the concept of unicode system ? What will we learn in Basics of Java ? History of Java Features of Java Hello Java Program Program Internal How to set path? Difference between JDK,JRE and JVM Internal Details of JVM Variable and Data Type Unicode System Operators |
| 16 | 4 | Java - Overview Java programming language was originally developed by Sun Microsystems which was initiated by James Gosling and released in 1995 as core component of Sun Microsystems' Java platform (Java 1.0 [J2SE]). As of December 2008, the latest release of the Java Standard Edition is 6 (J2SE). With the advancement of Java and its widespread popularity, multiple configurations were built to suite various types of platforms. Ex: J2EE for Enterprise Applications, J2ME for Mobile Applications. Sun Microsystems has renamed the new J2 versions as Java SE, Java EE and Java ME respectively. Java is guaranteed to be Write Once, Run Anywhere. Java is: Object Oriented: In Java, everything is an Object. Java can be easily extended since it is based on the Object model. Platform independent: Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by virtual Machine (JVM) on whichever platform it is being run. Simple:Java is designed to be easy to learn. If you understand the basic concept of OOP Java would be easy to master. Secure: With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption. Architectural-neutral :Java compiler generates an architecture-neutral object file format which makes the compiled code to be executable on many processors, with the presence of Java runtime system. Portable:Being architectural-neutral and having no implementation dependent aspects of the specification makes Java portable. Compiler in Java is written in ANSI C with a clean portability boundary which is a POSIX subset. Robust:Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking. Multithreaded: With Java's multithreaded feature it is possible to write programs that can do many tasks simultaneously. This design feature allows developers to construct smoothly running interactive applications. Interpreted:Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light weight process. High Performance: With the use of Just-In-Time compilers, Java enables high performance. Distributed:Java is designed for the distributed environment of the internet. Dynamic: Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time. History of Java: James Gosling initiated the Java language project in June 1991 for use in one of his many set-top box projects. The language, initially called Oak after an oak tree that stood outside Gosling's office, also went by the name Green and ended up later being renamed as Java, from a list of random words. Sun released the first public implementation as Java 1.0 in 1995. It promised Write Once, Run Anywhere (WORA), providing no-cost run-times on popular platforms. On 13 November 2006, Sun released much of Java as free and open source software under the terms of the GNU General Public License (GPL). On 8 May 2007, Sun finished the process, making all of Java's core code free and open-source, aside from a small portion of code to which Sun did not hold the copyright. Tools you will need: For performing the examples discussed in this tutorial, you will need a Pentium 200-MHz computer with a minimum of 64 MB of RAM (128 MB of RAM recommended). You also will need the following softwares: Linux 7.1 or Windows 95/98/2000/XP operating system. Java JDK 5 Microsoft Notepad or any other text editor This tutorial will provide the necessary skills to create GUI, networking, and Web applications using Java. Try It Option: We have provided you an option to compile and execute available code online. Just click on Try it button avaiable at top-right corner of the code window to compile and execute available code. There are certain examples which can not be executed online, so we have skipped those examples. public class MyFirstJavaProgram { public static void main(String []args) { System.out.println("Hello World"); } } There may be a case that you do not see the result of the compiled/executed code, in such case you can re-try to compile and execute the code using execute button available in compliation pop-up window. What is Next? Next chapter will guide you to where you can obtain Java and its documentation. Finally, it instructs you on how to install Java and prepare an environment to develop Java applications. |