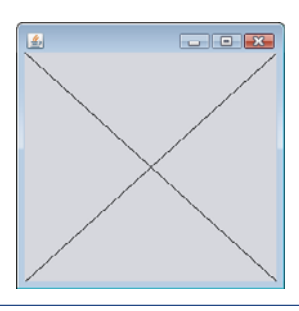
* **Android—the fastest growing mobile and smartphone operating system—is based on the Linux kernel and Java.**
* **One benefit of developing Android apps is the openness of the platform. The operating system is open source and free.**
* **You can place your apps on the online Android Market (**[**www.market.android.com**](http://www.market.android.com)**)**
* **Fortran Fortran (FORmula TRANslator) for scientific and engineering applications that require complex mathematical computations. It’s still widely used and its latest versions are object oriented.**
* **COBOL COBOL (COmmon Business Oriented Language) is still widely used for commercial applications that require precise and efficient manipulation of large amounts of data. Its latest version supports object-oriented programming**
* **Today, most of the code for general-purpose operating systems is written in C or C++**
* **C# (based on C++ and Java)**
* **System.out.printf( "%s\n%s\n", "Welcome to","Java Programming!");**
* **Common programming error: Placing a semicolon immediately after the right parenthesis of the condition in anifstatement is normally a logic error**
* **A Java class name is an identifier—a series of characters consisting of letters, digits, underscores (\_) and dollar signs ($) that does not begin with a digit and does not contain spaces.**
* **Java provides two primitive types for storing floating-point numbers in memory—float and double. They differ primarily in that double variables can store numbers with larger magnitude and finer detail**
* **Type double is preferred over type float ,because double variables can represent floating-point numbers more accurately**
* **Bohm and Jacopini’s work demonstrated that all programs could be written in terms of only three control structures—the sequence structure, the selection structure and the repetition structure**
* **Some programmers do not use program development tools like Pseudocode. They feel that their ultimate goal is to solve the problem on a computer and that writing Pseudocode merely delays the production of final outputs. Although this may work for simple and familiar problems, it can lead to serious errors and delays in large, complex projects.**
* **In a sentinel-controlled loop, prompts should remind the user of the sentinel. Sentinel is the input that will make the program terminate**
* **A cast operator can be used to convert between primitive numeric types, such as int and double, and between related reference types with Polymorphism. Casting to the wrong type may cause compilation errors or runtime errors.**
* **Every JPanel has a paint Component method, which the system automatically calls every time it needs to display the JPanel.**
* **Methods getWidth and getHeight return the JPanel’s width and height, respectively**
* **If you resize the window, the lines will scale accordingly, because the arguments are based on the width and height of the panel. Resizing the window in this application causes the system to call paintComponent to redraw the DrawPanel’s contents.**

****

* **This class will call the paintcompnent method automatically when it need to draw the content of it**

public class draw extends JPanel{

@Override

public void paintComponent(Graphics g)

{

g.drawLine(100,getX(),getWidth(),getHeight() );

}}

* **Because floating-point values may be approximate, controlling loops with floating-point variables may result in imprecise counter values and inaccurate termination tests.** **Use integers to control counting loops.**
* **Placing a semicolon immediately to the right of the right parenthesis of a for header makes that for’s body an empty statement. This is normally a logic error.**
* System.out.printf("%s%20s \n", "Year", "Amount on deposit");

**Or**

System.out.printf("%4d%,20.2f\n", year, amount );

**The integer 20 between % and the conversion character indicates that the value should be displayed with a field width of 20, the value is right justified in the field by default,**

**Output example: good work man man**

**To output values left justified, simply precede the field width with the minus sign (–) formatting flag(e.g.,%-20s)**

**Output example: good work man man**

* **If the variable is bigger than the specified length the field width would be extended to the right to accommodate the entire value—this would push the amount field to the right, upsetting the neat columns of our tabular output.**
* **Formatting Floating-Point Numbers: The comma (,) formatting flag indicates that the floating-point value should be output with a grouping separator. The.2specifies the formatted number’s precision—in this case, the number is rounded to the nearest hundredth and output with two digits to the right of the decimal point.**
* **you’ll learn how to use integers to perform precise monetary calculations. Java also provides class java.math.BigDecimalto perform precise monetary calculations**
* **Although each case and the default case in a switch can occur in any order, place the default case last. When the default case is listed last, the break for that case is not required.**
* **First, make your code simple and correct; then make it fast and small, but only if necessary.**
* **Short-Circuit Evaluation of Complex Conditions:**

**The parts of an expression containing&&or||operators are evaluated only until it’s known whether the condition is true or false.**

( gender == FEMALE) && ( age >=65 )

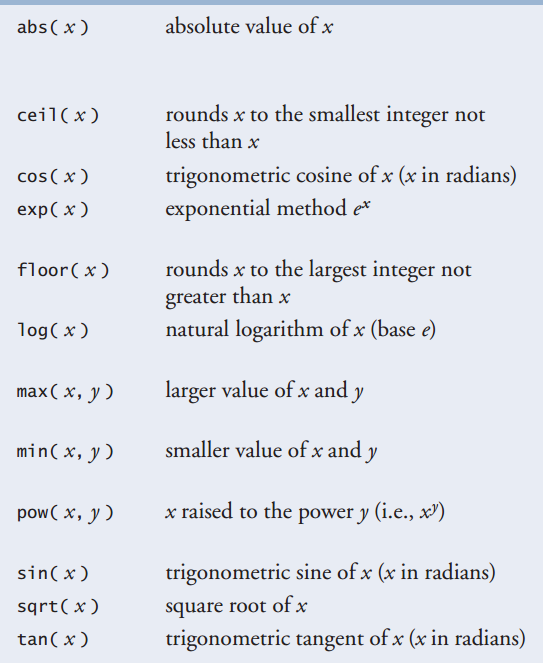
**Thus, evaluation of the expression stops immediately if gender is not equal to FEMALE**

* **In AND logic operator: For example, in the expression (i!=0)&&(10/i==2), the second condition must appear after the first condition, or a divide-by-zero error might occur.**
* **The boolean logical AND(&) and boolean logical inclusive OR(|)operators are identical to the&&and||operators, except that the & and | operators alwayse valuate both of their operands (i.e., they do not perform short-circuit evaluation).**

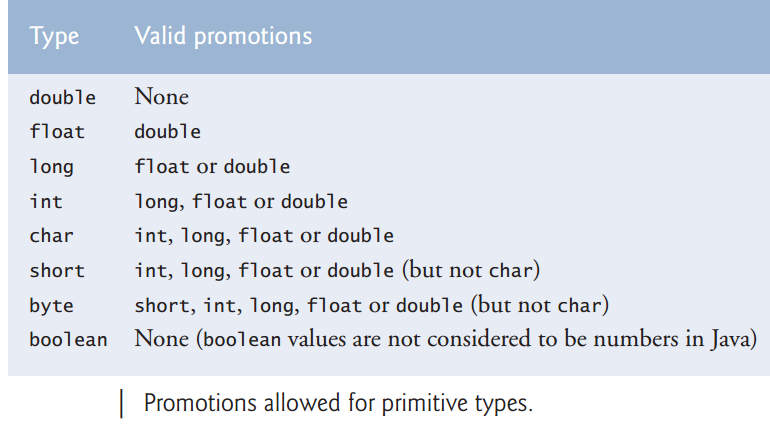
( gender == 1)&(age>=65 )

**So, the expression evaluates age>=65regardlessof whether gender is equal to 1.** **This is useful if the right operand of the boolean logical AND or boolean logical inclusive OR operator has a required side effect—a modification of a variable’s value**

* **Notice that & and | can work as bitwise operators**
* **A simple condition containing the boolean logical exclusive OR(^)operator is true if and only if one of its operands is true and the other is false**
* **Sometimes you have to use nested for loops that tries all possibilities. This method is an example of “brute-force” computing. You’ll learn in more advanced computer science courses that for many interesting problems there’s no known algorithmic approach other than using sheer brute force.**
* **DeMorgan’slawscansometimesmakeitmoreconvenientforustoexpressalogicalexpression. These laws state that !(condition1&&condition2) is logically equivalent to the expression (!condition1 || !condition2).the expression !(condition1||condition2)is logically equivalent to the expression (!condition1&& !condition2)**
* **To promote software reusability, every method should be limited to performing a single, well-defined task, and the name of the method should express that task effectively.**

****

* **Math.E (2.718281828459045) is the base value for natural logarithms (calculated with static Math method log)**
* **Methods can return at most one value, but the returned value could be a reference to an object that contains many values.**
* **A static method can call only other static methods of the same class directly (i.e., using the method name by itself) and can manipulate only static variables in the same class directly. To access the class’s non-static members, a static method must use a reference to an object of the class.**
* **Of course, a computer’s memory is finite, so only a certain amount can be used to store activation records on the program-execution stack. If more method calls occur than can have their activation records stored, an error known as a stack overflow occurs.**
* **To keep data safe while casting different data type you must covert to a larger data type**

****

* **Generalization for shifting and scaling random variable:**

**number = shiftingValue +**

**differenceBetweenValues \* randomNumbers.nextInt(scalingFactor );**

**EX: number = 2+ 3\* randomNumbers.nextInt( 5);**

* **Random-Number Repeatability for Testing and Debugging:**

**Class Random’s methods actually generate pseudorandom numbers based on complex mathematical calculations—the sequence of numbers appears to be random. The calculation that produces the numbers uses the time of day as a seed value to change the sequence’s starting point. Each new Random object seeds itself with a value based on the computer system’s clock at the time the object is created, enabling each execution of a program to produce a different sequence of random numbers.**

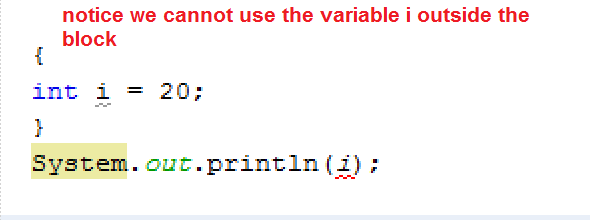
* **When debugging an application, it’s often useful to repeat the exact same sequence of pseudorandom numbers during each execution of the program. This repeatability enables you to prove that your application is working for a specific sequence of random numbers before you test it with different sequences of random numbers. When repeatability is important, you can create a Random object as follows:**
* Random randomNumbers =new Random( seedValue );

**Or**

* randomNumbers.set( seedValue );

**The seed Value argument (of type long) seeds the random-number calculation. If the same seed Value is used every time, the Random object produces the same sequence of numbers**

* **It’s a convention to use only uppercase letters in the names of enumeration constants. This makes them stand out and reminds you that they are not variables**
* **Using enumeration constants (like Status.WON, Status.LOST and Status.CONTINUE) rather than literal values (such as 0, 1 and 2)makes programs easier to read and maintain.**
* **Any block may contain variable declarations. If a local variable or parameter in a method has the same name as a field of the class, the field is “hidden” until the block terminates execution—this is called shadowing.**
* **Any variable defined inside braces cannot be shown outside this braces even if the braces are not for class or method**
* **EX:**

****

* **Declaring overloaded methods with identical parameter lists is a compilation error regardless of whether the return types are different.**
* **To set the color in GUI applications:**
* Color x = new Color(220,110, 0);

**Or**

* Color.getHSBColor(220, 110, 200)
* **A called method must know how to return to its caller, so the return address of the calling method is pushed onto the program-execution stack when the method is called. If a series of method calls occurs, the successive return addresses are pushed onto the stack in last-in, first-out order so that the last method to execute will be the first to return to its caller.**
* **(Rounding Numbers)To round numbers to specific decimal places, use a statement like**
* **y = Math.floor( x \* 10 +0.5 )/10;**

**which rounds x to the tenths position (i.e., the first position to the right of the decimal point), or**

* **y = Math.floor( x \* 100 + 0.5)/100;**

**which rounds x to the hundredths position (i.e., the second position to the right of the decimal point**

* **An index must be an int value or a value of a type that can be promoted to int —namely, byte , short or char, but not long; otherwise, a compilation error occurs.**
* **In an array declaration, specifying the number of elements in the square brackets of the declaration (e.g.,int[12] c;) is a syntax error.**
* **Declaring multiple array variables in a single declaration can lead to subtle errors. Consider the declaration int[] a, b, c;. If a, b and c should be declared as array variables, then this declaration is correct—placing square brackets directly following the type indicates that all the identifiers in the declaration are array variables. However, if only a is intended to be an array variable, and b and c are intended to be individual int variables, then this declaration is incorrect—the declaration int a[], b, c; would achieve the desired result.**
* **Good idea: we can use the Elements of an Array as Counter.**
* **Good idea: we can use Arrays to Analyze Survey Results like using the content of the array as an index to anther array.**
* **The enhanced for statement simplifies the code for iterating through an array. Note, however, that the enhanced for statement can be used only to obtain array elements—it cannot be used to modify elements. If your program needs to modify elements, use the traditional counter-controlled for statement**
* **A multidimensional array in which each row has a different number of columns can be created as follows:**

int[][] b =new int[ 2][ ]; // create 2 rows

b[ 0]=new int[5];// create 5 columns for row 0

b[ 1]=new int[3]; // create 3 columns for row 1

**The preceding statements create a two-dimensional array with two rows. Row 0 has five columns, and row1has three columns.**

* **Placing an ellipsis indicating a variable-length argument list in the middle of a parameter list is a syntax error. An ellipsis may be placed only at the end of the parameter list.**
* **Class Arrays helps you avoid reinventing the wheel by providing static methods for common array manipulations.**
* **The first argument (intArray) passed to System method arraycopy is the array from which elements are to be copied. The second argument (0) is the index that specifies the starting point in the range of elements to copy from the array.This value can be any valid array index. The third argument (intArrayCopy) specifies the destination array that will store the copy. The fourth argument (0) specifies the index in the destination array where the first copied element should be stored. The last argument specifies the number of elements to copy from the array in the first argument. In this case, we copy all the elements in the array.**

System.arraycopy( intArray, 0, intArrayCopy,0, intArray.length );

* **Passing an unsorted array to binarySearch is a logic error—the value returned is undefined.**
* **The capacity indicates how many items the ArrayList can hold without growing. ArrayList is implemented using an array behind the scenes. When the ArrayList grows, it must create a larger internal array and copy each element to the new array. This is a time-consuming operation. It would be inefficient for the ArrayList to grow each time an element is added. Instead, it grows only when an element is added and the number of elements is equal to the capacity.**
* **ArrayList has a method called trimToSize.it trims the capacity of the ArrayList to current number of elements.**
* **When implementing a method of a class, use the class’s set and get methods to access the class’s private data. This simplifies code maintenance and reduces the likelihood of errors.**
* **Ensure that you do not include a return type in a constructor definition. Java allows other methods of the class besides its constructors to have the same name as the class and to specify return types. Such methods are not constructors and will not be called when an object of the class is instantiated**
* **A public set method can—and should—carefully scrutinize attempts to modify the variable’s value and throw an exception if necessary. For example, an attempt to set the day of the month to 37 would be rejected, an attempt to set a person’s weight to a negative value would be rejected, and so on. Thus, although set and get methods provide access to private data, the access is restricted by the implementation of the methods. This helps promote good software engineering.**
* **A class can have references to objects of other classes as members. This is called composition and is sometimes referred to as a has-a-relationship**
* **In an enum declaration, it’s a syntax error to declare enum constants after the enum type’s constructors, fields and methods.**
* **Every object uses system resources, such as memory. We need a disciplined way to give resources back to the system when they’re no longer needed; otherwise, “resource leaks” might occur that would prevent them from being reused by your program or possibly by other programs. The JVM performs automatic garbage collection to reclaim the memory occupied by objects that are no longer used. When there are no more references to an object, the object is eligible to be collected. This typically occurs when the JVM executes its garbage collector. So, memory leaks that are common in other languages like C and C++ (because memory is not automatically reclaimed in those languages) are less likely in Java, but some can still happen in subtle ways. Other types of resource leaks can occur.For example, an application may open a fileon disk to modify its contents. If it does not close the file, the application must terminate before any other application can use it.**
* **The finalize method is called by the garbage collector to perform termination housekeeping on an object just before the garbage collector reclaims the object’s memory. Method finalize does not take parameters and has return type void .A problem with method finalize is that the garbage collector is not guaranteed to execute at a specified time. In fact, the garbage collector may never execute before a program terminates. Thus, it’s unclear whether, or when, method finalize will be called. For this reason, most programmers should avoid method finalize.**
* **A class that uses system resources, such as files on disk, should provide a method that programmers can call to release resources when they’re no longer needed in a program. Many Java API classes provide close or dispose methods for this purpose. For example, class Scanner has a close method.**
* **Invoke every static method by using the class name and a dot (.) to emphasize that the method being called is a static method**
* **A static import declaration has two forms—one that imports a particular static member (which is known as single static import) and one that imports all static members of a class (known as static import on demand). The following syntax imports a particular static member:**

import static packageNme.ClassName.staticMemberName;

**where packageName is the package of the class (e.g.,java.lang),ClassNameis the name of the class (e.g., Math)and static MemberName is the name of the static field or method (e.g.,PIorabs). The following syntax imports all static members of a class:**

import static packageName.ClassName.\*;

**The asterisk (\*) indicates that all static members of the specified class should be available for use in the file. static import declarations import only static class members. Regular import statements should be used to specify the classes used in a program.**

* **A compilation error occurs if a program attempts to import two or more classes’ static methods that have the same signature or static fields that have the same name.**
* **A final field should also be declared static if it’s initialized in its declaration to a value that’s the same for all objects of the class. After this initialization, its value can never change. Therefore, we don’t need a separate copy of the field for every object of the class. Making the field static enables all objects of the class to share the final field.**
* **a final(constant). Such variables can be initialized when they’re declared. If they are not, they must be initialized in every constructor of the class. Initializing constants in constructors enables each object of the class to have a different value for the constant. If a final variable is not initialized in its declaration or in every constructor, a compilation error occurs.**
* **Using the import declaration** import java.\*; **causes a compilation error. You must specify the exact name of the package from which you want to import classes.**
* **When the compiler encounters a method declared with @Override, it compares the method’s signature with the superclass’s method signatures. If there isn’t an exact match, the compiler issues an error message, such as “method does not override or implement a method from a super type.” This indicates that you’ve accidentally overloaded a superclass method. You can then fix your method’s signature so that it matches one in the superclass.**
* **It’s a syntax error to override a method with a more restricted access modifier—a public method of the superclass cannot become a protected or private method in the subclass; a protected method of the superclass cannot become a private method in the subclass. Doing so would break the is- a relationship in which it’s required that all subclass objects be able to respond to method calls that are made to public methods declared in the superclass. If a public method, for example, could be overridden as a protected or private method, the subclass objects would not be able to respond to the same method calls as superclass objects. Once a method is declared public in a superclass, the method remains public for all that class’s direct and indirect subclasses.**
* **With inheritance, the common instance variables and methods of all the classes in the hierarchy are declared in a superclass. When changes are made for these common features in the superclass—subclasses then inherit the changes. Without inheritance, changes would need to be made to all the source-code files that contain a copy of the code in question.**
* **Declaring superclass instance variables private(as opposed to protected) enables the superclass implementation of these instance variables to change without affecting subclass implementations.**
* **When a superclass method is overridden in a subclass, the subclass version often calls the superclass version to do a portion of the work. Failure to prefix the superclass method name with the keyword super and a dot (.) separator when calling the superclass’s method causes the subclass method to call itself, potentially creating an error called infinite recursion**
* **Java ensures that even if a constructor does not assign a value to an instance variable, the variable is still initialized to its default value (e.g.,0for primitive numeric types, false for Booleans ,null for references).**
* **Although inheriting from a class does not require access to the class’s source code, developers often insist on seeing the source code to understand how the class is implemented. Developers in industry want to ensure that they’re extending a solid class**
* **We can display text and ImageIcon on a JLabel but next to each other**

ImageIcon labelIcon =newImageIcon( "GUItip.gif");

JLabel southLabel = new JLabel( labelIcon );

southLabel.setText( "South");

* **We can define the position in the Jframe using layout**

application.add( northLabel, BorderLayout.NORTH);

* **An is-arelationship represents inheritance. In an is-arelationship, an object of a subclass also can be treated as an object of its superclass.**
* **A has-arelationship represents composition. In a has-arelationship, a class object contains references to objects of other classes.**
* **Polymorphism promotes extensibility: Software that invokes polymorphic behavior is independent of the object types to which messages are sent. New object types that can respond to existing method calls can be incorporated into a system without modifying the base system. Only client code that instantiates new objects must be modified to accommodate new types.**
* **An abstract class declares common attributes and behaviors (both abstract and concrete) of the various classes in a class hierarchy. An abstract class typically contains one or more abstract methods that subclasses must override if they are to be concrete. The instance variables and concrete methods of an abstract class are subject to the normal rules of inheritance**
* **Failure to implement a superclass’s abstract methods in a subclass is a compilation error unless the subclass is also declared abstract.**
* **Although we cannot instantiate objects of abstract super classes, you’ll soon see that we can use abstract super classes to declare variables that can hold references to objects of any concrete class derived from those abstract super classes. Programs typically use such variables to manipulate subclass objects polymorphically. You also can use abstract superclass names to invoke static methods declared in those abstract super classes.**
* **All calls to methods are resolved at execution time, based on the type of the object to which subClassObject refers. This process is known as dynamic binding or late binding**
* **Assigning a superclass variable to a subclass variable (without an explicit cast) is a compilation error.**
* **If a subclass object’s reference has been assigned to a variable of one of its direct or indirect Super classes at execution time, it’s acceptable to downcast the reference stored in that superclass variable back to a subclass-type reference. Before performing such a cast, use the instanceof operator to ensure that the object is indeed an object of an appropriate subclass.**
* **When downcasting a reference, a ClassCast Exception occurs if the referenced object at execution time does not have an is-arelationship with the type specified in the cast operator.**
* **1. Assigning a superclass reference to a superclass variable is straightforward.**

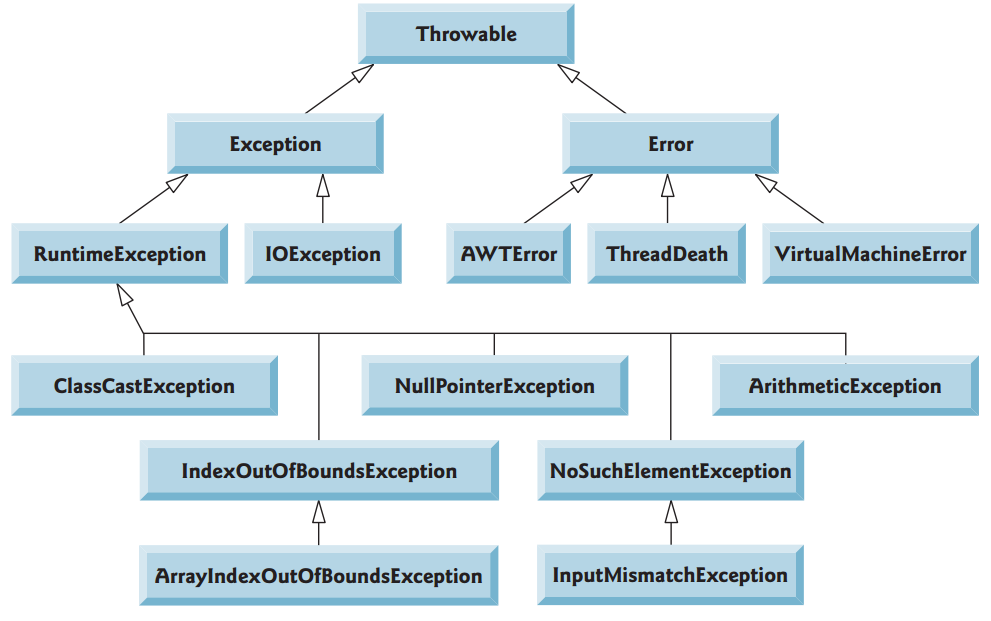
**2. Assigning a subclass reference to a subclass variable is straightforward.**

**3. Assigning a subclass reference to a superclass variable is safe, because the subclass object is an object of its superclass. However, the superclass variable can be used to refer only to superclass members. If this code refers to subclass-only members through the superclass variable, the compiler reports errors.**

**4. Attempting to assign a superclass reference to a subclass variable is a compilation error. To avoid this error, the superclass reference must be cast to a subclass type explicitly. At execution time, if the object to which the reference refers is not a subclass object, an exception will occur. (For more on exception handling, see Chapter 11.) You should use the instanceof operator to ensure that such a cast is performed only if the object is a subclass object.**

* **When declaring a method in an interface, choose a method name that describes the method’s purpose in a general manner, because the method may be implemented by many unrelated classes**
* **All objects of a class that implement multiple interfaces have the is-a relationship with each implemented interface type**
* **Some good java API interfaces: Runnable, Comparable, Serializable**
* **This information is known as a stack trace, which includes the name of the exception ( java.lang.ArithmeticException ) in a descriptive message that indicates the problem that occurred and the method-call stack (i.e., the call chain) at the time it occurred.**
* **. A catch block(also called a catch clause or exception handler) catches (i.e., receives) and handles an exception.**
* **At least one catch block or a finally block(discussed in Section 11.6) must immediately follow the try block**
* **Each catch block can have only a single parameter—specifying a comma-separated list of exception parameters is a syntax error.**
* **An uncaught exception is one for which there are no matching catch blocks. You saw uncaught exceptions in the second and third outputs of Fig. 11.1. Recall that when exceptions occurred in that example, the application terminated early (after displaying the exception’s stack trace). This does not always occur as a result of uncaught exceptions. Java uses a “multithreaded” model of program execution—each thread is a parallel activity. One program can have many threads. If a program has only one thread, an uncaught exception will cause the program to terminate. If a program has multiple threads, an uncaught exception will terminate only the thread where the exception occurred. In such programs, however, certain threads may rely on others, and if one thread terminates due to an uncaught exception, there may be adverse effects to the rest of the program. Chapter 26, Multithreading, discusses these issues in depth.**
* **After the exception is handled, program control does not return to the throw point, because the try block has expired(and its local variables have been lost). Rather, control resumes after the last catch block. This is known as the termination model of exception handling. Some languages use the resumption model of exception handling, in which, after an exception is handled, control resumes just after the throw point.**
* **Read the online API documentation for a method before using it in a program. The documentation specifies the exceptions thrown by the method (if any) and indicates reasons why such exceptions may occur. Next, read the online API documentation for the specified exception classes. The documentation for an exception class typically contains potential reasons that such exceptions occur. Finally, provide for handling those exceptions in your program.**
* **Incorporate your exception-handling strategy into your system from the inception of the design process. Including exception handling after a system has been implemented can be difficult.**
* **The difference between Exceptions and Errors:**

**Errors happen infrequently and should not be caught by applications—it’s usually not possible for applications to recover from Errors.**

****

* **Java distinguishes between checked exceptions and unchecked exceptions. This distinction is important, because the Java compiler enforces a catch-or-declare requirement for checked exceptions. An exception’s type determines whether it’s checked or unchecked. All exception types that are direct or indirect subclasses of class Runtime Exception (packagejava.lang) are unchecked exceptions. These are typically caused by defects in your program’s code. Examples of unchecked exceptions include ArrayIndexOutOfBounds Exceptions and Arithmetic Exceptions. All classes that inherit from Class Exception but not class Runtime Exception are considered to be checked exceptions. Such exceptions are typically caused by conditions that are not under the control of the program—for example, in file processing, the program can’t open a file because the file does not exist. Classes that inherit from class Error are considered to be unchecked.**
* **The compiler checks each method call and method declaration to determine whether the method throws checked exceptions. If so, the compiler verifies that the checked exception is caught or is declared in a throws clause.**
* **If a subclass method overrides a superclass method, it’s an error for the subclass method to list more exceptions in its throws clause than the overridden superclass method does. However, a subclass’s throws clause can contain a subset of a superclass’s throws list.**
* **Unlike checked exceptions, the Java compiler does not check the code to determine whether an unchecked exception is caught or declared. Unchecked exceptions typically can be prevented by proper coding. For example, the unchecked ArithmeticException thrown by method quotient can be avoided if the method ensures that the denominator is not zero before attempting to perform the division. Unchecked exceptions are not required to be listed in a method’s throws clause—even if they are, it’s not required that such exceptions be caught by an application.**
* **If there are multiple catch blocks that match a particular exception type, only the first matching catch block executes when an exception of that type occurs. It’s a compilation error to catch the exact same type in two different catch blocks associated with a particular try block. However, there may be several catch blocks that match an exception.**
* **Java performs automatic garbage collection of memory no longer used by programs, thus avoiding most memory leaks. However, other types of resource leaks can occur. For example, files, database connections and network connections that are not closed properly after they’re no longer needed might not be available for use in other programs.**
* **A subtle issue is that Java does not entirely eliminate memory leaks. Java will not garbage collect an object until there are no remaining references to it. Thus, if you erroneously keep references to unwanted objects, memory leaks can occur. To help avoid this problem, set reference-type variables to null when they’re no longer needed.**
* **The finally block will execute whether or not an exception is thrown in the corresponding try block. The finally block also will execute if a try block exits by using a return, break or continue statement or simply by reaching its closing right brace. The finally block will not execute if the application exits early from a try block by calling method System.exit.**
* **The finally block is an ideal place to release resources acquired in a try block (such as opened files), which helps eliminate resource leaks.**
* **System.out and System.err are streams—sequences of bytes. While System.out (known as the standard output stream) displays a program’s output, System.err (known as the standard error stream) displays a program’s errors. Output from these streams can be redirected (i.e., sent to somewhere other than the command prompt, such as to a file). Using two different streams enables you to easily separate error messages from other output. For instance, data output from System.err could be sent to a log file, while data output fromSystem.out can be displayed on the screen.**
* **When toString is invoked on any Throwable object, its resulting string includes the descriptive string that was supplied to the constructor, or simply the class name if no string was supplied.**
* **Exceptions can be thrown from constructors. When an error is detected in a constructor, an exception should be thrown to avoid creating an improperly formed object.**
* **Exceptions are rethrown when a catch block, upon receiving an exception, decides either that it cannot process that exception or that it can only partially process it. Rethrowing an exception defers the exception handling (or perhaps a portion of it) to another catch block associated with an outer try statement. An exception is rethrown by using the throw keyword, followed by a reference to the exception object that was just caught. Exceptions cannot be rethrown from a finally block, as the exception parameter (a local variable) from the catch block no longer exists.**
* **If an exception has not been caught when control enters afinallyblock and the finally block throws an exception that’s not caught in the finally block, the first exception will be lost and the exception from the finally block will be returned to the calling method.**
* **Assuming that an exception thrown from a catch block will be processed by that catch block or any other catch block associated with the same try statement can lead to logic errors.**
* **Exception handling is intended to remove error-processing code from the main line of a program’s code to improve program clarity. Do not place try…catch…finally around every statement that may throw an exception. This makes programs difficult to read. Rather, place one try block around a significant portion of your code, follow that try block with catch blocks that handle each possible exception and follow the catch blocks with a single finally block (if one is required).**
* **An exception that’s not caught in an application causes Java’s default exception handler to run. This displays the name of the exception, a descriptive message that indicates the problem that occurred and a complete execution stack trace. In an application with a single thread of execution, the application terminates. In an application with multiple threads, the thread that caused the exception terminates.**
* **Chained Exceptions: Sometimes a method responds to an exception by throwing a different exception type that’s specific to the current application. If a catch block throws a new exception, the original exception’s information and stack trace are lost.**

**We can solve it using Exception constructor with two arguments so it will keep the old exceptions and add new one to them**

throw new Exception( "Exception thrown in method1", exception );

* **If possible, indicate exceptions from your methods by using existing exception classes, rather than creating new ones. The Java API contains many exception classes that might be suitable for the type of problems your methods need to indicate.**
* **A new exception class must extend an existing exception class to ensure that the class can be used with the exception-handling mechanism. Like any other class, an exception class can contain fields and methods. A typical new exception class contains only four constructors: one that takes no arguments and passes a default error message String to the superclass constructor; one that receives a customized error message as a String and passes it to the superclass constructor; one that receives a customized error message as a String and a Throwable (for chaining exceptions) and passes both to the super class constructor; and one that receives a Throwable (for chaining exceptions) and passes it to the superclass constructor.**
* **When defining your own exception type, study the existing exception classes in the Java API and try to extend a related exception class. For example, if you’re creating a new class to represent when a method attempts a division by zero, you might extend class ArithmeticException because division by zero occurs during arithmetic. If the existing classes are not appropriate super classes for your new exception class, decide whether your new class should be a checked or an unchecked exception class. The new exception class should be a checked exception (i.e., extend Exception but not RuntimeException )if clients should be required to handle the exception. The client application should be able to reasonably recover from such an exception. The new exception class should extend RuntimeException if the client code should be able to ignore the exception (i.e., the exception is an unchecked one).**
* **By convention, all exception-class names should end with the word Exception.**
* **The assert statement evaluates a Boolean expression and, if false, throws an Assertion Error (a subclass of Error).**
* **The first form of the assert statement which throws an AssertionError if expression is false.**

assert expression ;

* **The second form which evaluatesexpression1and throws an AssertionError with expression 2 as the error message if expression 1 is false.**

assert expression1 : expression2 ;

* **You use assertions primarily for debugging and identifying logic errors in an application. You must explicitly enable assertions when executing a program, because they reduce performance and are unnecessary for the program’s user.**
* **Multi- catch to handle multiple exceptions**

catch ( Type 1 | Type 2 | Type 3e)

* **Where ClassName is a class that implements the AutoCloseable interface. This code creates an object of type ClassName and uses it in the try block, then calls its close method to release any resources used by the object. The try-with-resources statement implicitly calls the theObject’s close method at the end of the try block. You can allocate multiple resources in the parentheses following try by separating them with a semicolon (;).**
* try (ClassName theObject = new ClassName() )

{ }

catch( Exception e )

{ }

* System.gc( ) **calls java to perform system garbage collection**
* System.out.println(new DecimalFormat("00.00").format(1222.32334));

**Is used to format a decimal number**

* **“is-a” vs. “has-a”**
* **“is-a”**

**Inheritance**

**subclass object treated as superclass object**

**Example: Car *is a* vehicle**

**Vehicle properties/behaviors also car properties/behaviors**

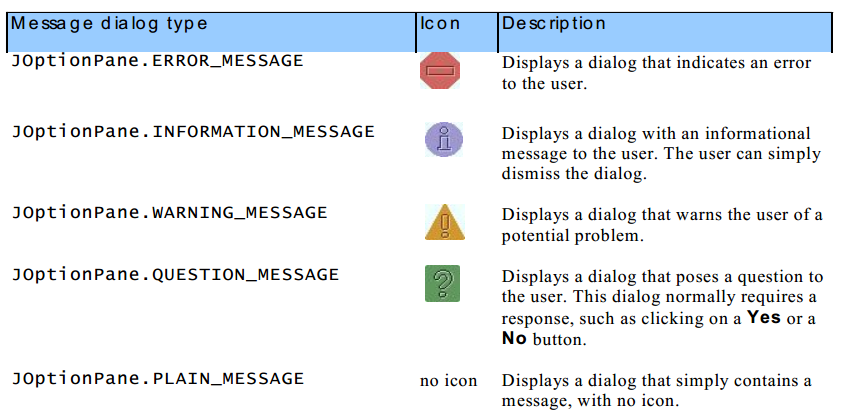
* **“has-a”**

**Composition**

**Object contains one or more objects of other classes as members**

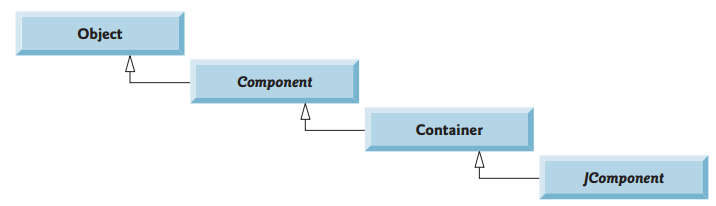
**Example: Car *has a* steering wheel**

* **For Joptionpane Message**

****

* **GUI components are sometimes called controls or widgets—short for window gadgets.**
* **There are actually two sets of Java GUI components. In Java’s early days, GUIs were built with components from the Abstract Window Toolkit (AWT) in package java.awt.**
* **Lightweight vs. Heavyweight GUI Components:**

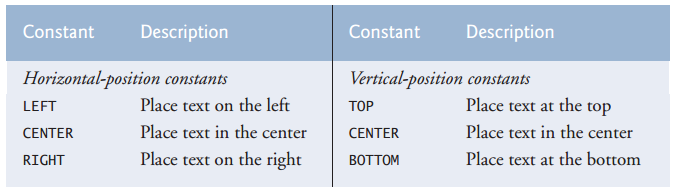
**Most Swing components are lightweight components—they’re written, manipulated and displayed completely in Java. AWT components are heavyweight components, because they rely on the local platform’s windowing system to determine their functionality and their look-and-feel. Several Swing components are heavyweight components.**

****

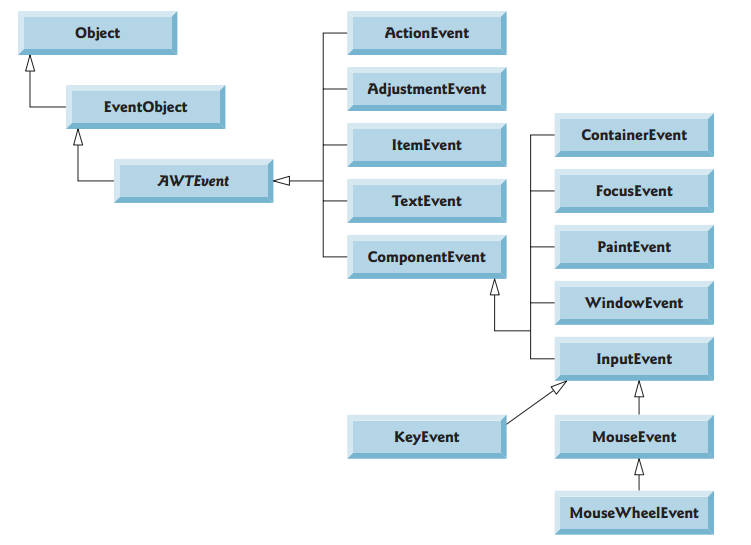
Swing class

ATM class

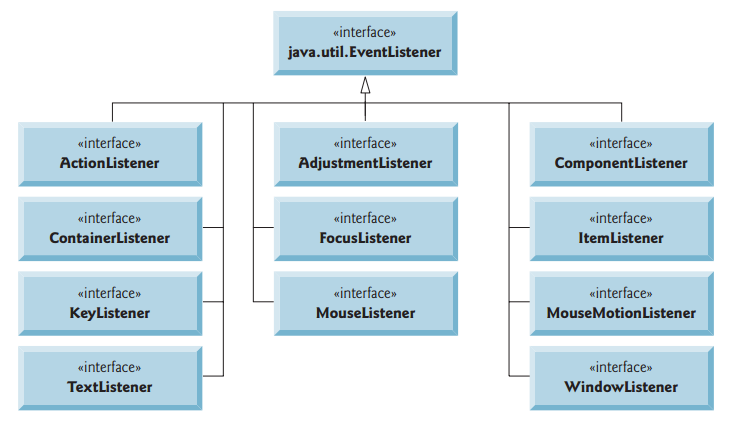
* **Java provides several layout managers that can help you position components**
* **We use Java’s layout managers to size and position components. With the FlowLayout layout manager, components are placed on a container from left to right in the order in which they’re added. When no more components can fit on the current line, they continue to display left to right on the next line. If the container is resized, a FlowLayout reflows the components, possibly with fewer or more rows based on the new container width.**
* **Like: mainFrm.setLayout(new FlowLayout());**
* **Method setToolTipText to specify the tool tip that’s displayed when the user positions the mouse cursor over the**
* **Like: Next.setToolTipText("woorking");**
* **the expressiongetClass().getResource("bug1.png")invokes method getClass to retrieve a reference to the Class object that represents the Label Frame class declaration. That reference is then used to invoke Class method getResource, which returns the location of the image as a URL**
* **like: Icon bug = newImageIcon( getClass().getResource( "bug1.png"));**
* **vertical and horizontal aligned methods used to define the position of the text in the label box**
* **label3.setHorizontalTextPosition( SwingConstants.CENTER);**
* **label3.setVerticalTextPosition(SwingConstants.BOTTOM );**
* **swing constants are integers used to indicated the value for the method to do the wanted operation**

****

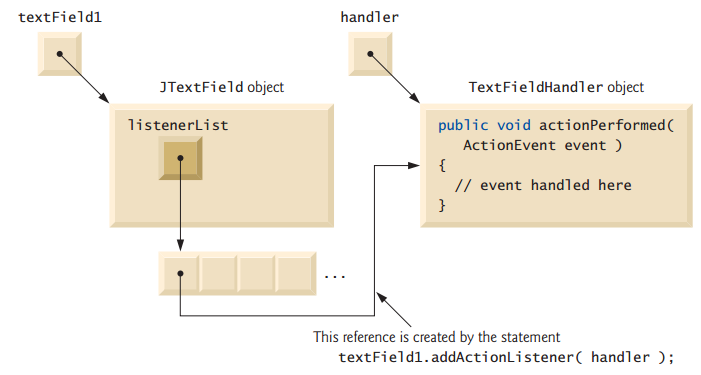
* **Jpassword field used to allow the user enter a password:**
* **JPasswordField p = new JPasswordField();**
* **You can type only in the text field that’s “in focus.” When you click a component, it receives the focus. This is important, because the text field with the focus is the one that generates an event when you press Enter.**
* **JPasswordField is a subclass of JTextArea**
* **// construct textfield with 10 columns**
* **textField1 = new JTextField(10);**
* **// construct textfield with default text and 21 columns**
* **textField3 = new JTextField("Uneditable text field",21 );**
* **Non-static nested classes are called inner classes and are frequently used to implement event handlers**
* **Normally, a component’s supported events are described in the Java API documentation for that component’s class and its superclasses.**

****

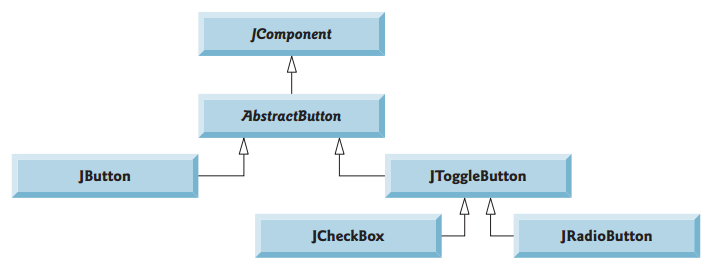
* **To know the object with which we have event**
* **event.getSource() == p**

****

* **Every JComponenthas an instance variable called listenerListthat refers to an object of class EventListenerList. Each object of a JComponent subclass maintains references to its registered listeners in the listenerList.**

****

* **Each event type has one or more corresponding event-listener interfaces. For example, ActionEventsare handled by ActionListeners, MouseEvents by MouseListeners and Mouse MotionListeners and KeyEvents by KeyListeners. When an event occurs, the GUI component receives (from the JVM) a unique event ID specifying the event type. The GUI component uses the event ID to decide the listener type to which the event should be dispatched and to decide which method to call on each listener objects.**

****

* **Using rollover icons for JButtons provides users with visual feedback indicating that when they click the mouse while the cursor is positioned over the JButton, an action will occur.**
* **To set the roll over image**
* **fancyJButton.setRolloverIcon( bug2 ); // set rollover image**
* **To refer to the top class object from an inner class object topClass.this and to make the message center of some place enter the component as an argument for it**
* **JOptionPane.showMessageDialog( , String.format(45 "You pressed: %s",));**
* **in that normally several JRadioButtons are grouped together and are mutually exclusive—only one in the group can be selected at any time**
* **// determine which CheckBoxes are checked and create Fon**

**if (checkName.isSelected )**

* **Notice that we can add elements in the font:**
* **newFont( "Serif" , Font.BOLD+Font.ITALIC , 14 );**
* **how to make an easy event:**
* **make inner class implements ActionListener**
* **set you code inside this class and you can specify some objects names**
* **define an object of the inner class in the top class**
* **add action listener with this inner class object to the compoenet**

**like:**

* **we use ButtonGroup to group radiobuttons in one group so only one of them is selected like:**
* **ButtonGroup g = new ButtonGroup();**
* **g.add(radioName);**
* **we don’t need to add the ButtonGroup object to the Frame but we need to add its elements to the Frame**
* **JRadioButtons, like JCheckBoxes, generateItemEvents when they’re clicked.**
* **A combo box (sometimes called a drop-down list) enables the user to select one item from a list**
* **In ComboBox c.setMaximumRowCount(3);**

**This method defines the maximum elements in the row when we active the ComboBox**

* **If we use FlowLayout in the frame we can use setSize() method for all components to be displayed**
* **An anonymous inner class declared in a method can access the instance variables and methods of the top-level class object that declared it, as well as the method’s final local variables, but cannot access the method’s non-final local variables.**
* **JList has some similar methods as ComboBox**
* **colorJList.setVisibleRowCount( 5); // display five rows at once**
* **// do not allow multiple selections**

**colorJList.setSelectionMode( ListSelectionModel.SINGLE\_SELECTION);**

* **Unlike a JComboBox, a JList does not provide a scrollbar if there are more items in the list than the number of visible rows. In this case, a JScrollPane object is used to provide the scrolling capability.**
* **To make JList scrollbar if there is more elements than specified in method setVisibleRowCount() so**
* **mainFrm.add(new Scrollpane(JListName));**
* **JLIst need to use method addListSelectionListener() and the overridden method is valueChanged()**
* **To change a background of a JFrame:**
* **mainFrm.getContentPane().setBackground(Color.white);**
* **Each JFrame actually consists of three layers—the background, the content pane and the glass pane. The content pane appears in front of the background and is where the GUI components in the JFrame are displayed. The glass pane is used to display tool tips and other items that should appear in front of the GUI components on the screen. The content pane completely hides the background of the JFrame; thus, to change the background color behind the GUI components, you must change the content pane’s background color. Method getContentPane() turns a reference to the JFrame’s content pane**
* **A multiple-selection list enables the user to select many items from a JList a SINGLE\_INTERVAL\_SELECTION list allows selecting a contiguous range of items. To do so, click the first item, then press and hold the Shift key while clicking the last item in the range. A MULTIPLE\_INTERVAL\_SELECTION list (the default) allows continuous range selection as described for a SINGLE\_INTERVAL\_SELECTION list. Such a list also allows miscellaneous items to be selected by pressing and holding the Ctrl key while clicking each item to select. To deselect an item, press and hold the Ctrl key while clicking the item a second time.**
* **Also we have:**
* **copyJList.setFixedCellWidth( 100); // set width**
* **copyJList.setFixedCellHeight( 15); // set height**
* **we can get the selected values by: colorJList.getSelectedValues();**
* **we can set the content of a List: copyJList.setListData();**
* **we can add listeners to Buttons by**

**copyJButton.addActionListener(**

**new ActionListener() // anonymous inner class**

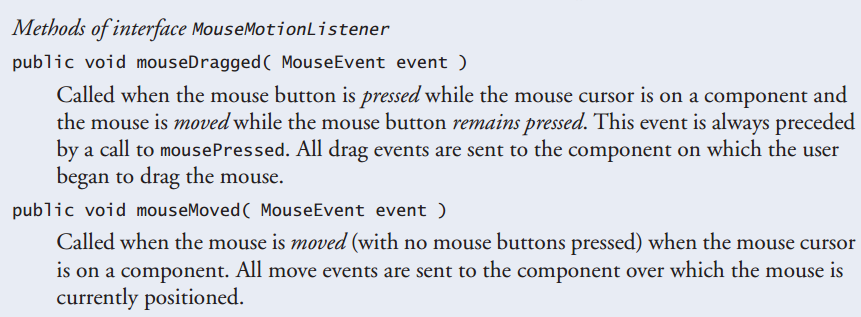
**{public void actionPerformed ( ActionEvent event )**

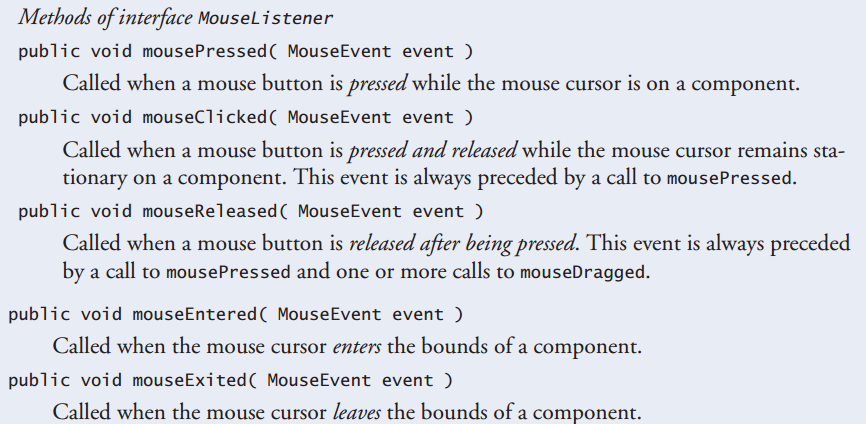
**{**

**}// end method actionPerformed**

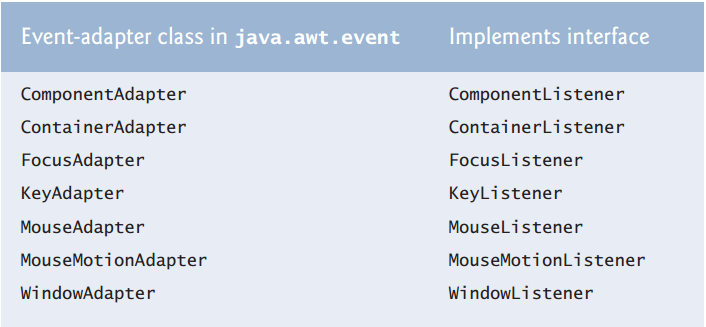
**}// end anonymous inner class**

**); // end call to addActionListener**

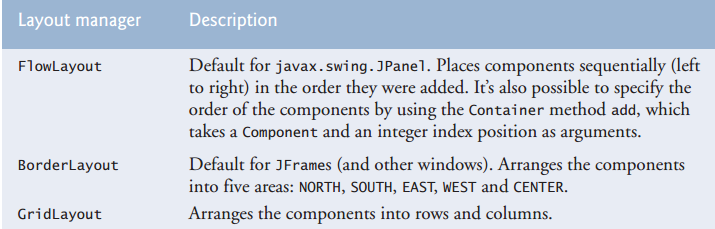
****

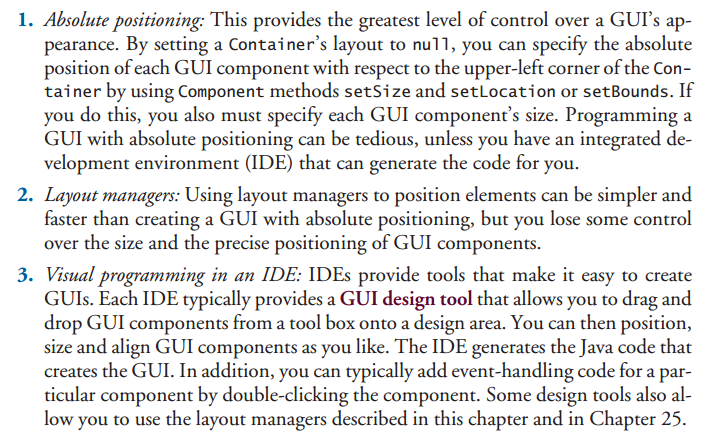
****

* **Calls to mouseDragged are sent to the MouseMotionListener for theComponenton which the drag started. Similarly, the mouseReleased call at the end of a drag operatio is sent to the MouseListener for the Component on which the drag operation started.**
* **Java also provides interface MouseWheelListener to enable applications to respond to the rotation of a mouse wheel. This interface declares method mouseWheelMoved, which receives a MouseWheel Event as its argument.**
* **We can define Jpanel to define a space we want to work act with mouse in it**
* **static JPanel jp = new JPanel();**

****

* **if we need to make event for only one method form a listener that has a lot of methods we make class extends this adapter of this listeners then we choose the wanted method to be overridden**
* **for mouse listener events we have some methods to get more information:**
* **number of clicks event.getClickCount()**
* **if (event.isMetaDown() )// right mouse button**
* **if (event.isAltDown() )// middle mouse button**
* **If you extend an adapter class and misspell the name of the method you’re overriding, your method simply becomes another method in the class. This is a logic error that is difficult to detect, since the program will call the empty version of the method inherited from the adapter class.**
* **To simulate a middle-mouse-button click on a one- or two-button mouse, the user can press the Alt key and click the only or left mouse button, respectively.**
* **In a JComponent subclass’s paintComponent method, the first statement should always call the superclass’s paint Component method to ensure that an object of the subclass displays correctly.**
* **If an overridden paintComponent method does not call the superclass’s version, the subclass component may not display properly. If an overridden paintComponent method calls the superclass’s version after other drawing is performed, the drawing will be erased.**
* **Method repaint to indicate that the PaintPanel should be refreshed on the screen as soon as possible with a call to the PaintPanel’s paintComponent method. Method paintComponent, which receives a Graphics parameter, is called automatically any time the PaintPanel needs to be displayed on the screen—such as when the GUI is first displayed—or refreshed on the screen—such as when method repaint is called or when the GUI component has been hidden by another window on the screen and subsequently becomes visible again. >>> super.paintComponent( g );// clears drawing area**
* **Calling repaint for a Swing GUI component indicates that the component should be refreshed on the screen as soon as possible. The component’s background is cleared only if the component is opaque. JComponent method setOpaque can be passed a Boolean argument indicating whether the component is opaque (true) or transparent (false).**
* **Point object is used to keep Cartesian coordinates x , y:**
* **Point p = event.getPoint( );**
* **Method keyTyped is called in response to pressing any key that is not an action key. (The action keys are any arrow key, Home, End, Page Up, Page Down, any function key, etc.) but keyPressed is for all keys.**
* **Method getKeyText, which returns a string containing the name of the key that was pressed**
* **Method getKeyChar (which returns a char) to get the Unicode value of the character typed**
* **Method isActionKey to determine whether the key in the event was an action key**
* **Method getKeyCode to get the virtual key code of the pressed key**
* **Method getModifiers is called (line 59) to determine whether any modifier keys (such as Shift, Alt and Ctrl) were pressed when the key event occurred. The result of this method is passed to static KeyEvent method getKeyModifiersText, which produces a string containing the names of the pressed modifier keys.**
* **A key constant for each one. These constants can be used from the key event handlers to determine whether a particular key was pressed. Also, to determine whether the Alt, Ctrl, Meta and Shift keys are pressed individually, InputEvent methods isAltDown, isControlDown, isMetaDownand isShiftDown each return a Boolean indicating whether the particular key was pressed during the key event.**
* **Layout managers arrange GUI components in a container for presentation purposes. You can use the layout managers for basic layout capabilities instead of determining every GUI component’s exact position and size. This functionality enables you to concentrate on the basic look-and-feel and lets the layout managers process most of the layout details. All layout managers implement the interface LayoutManager**

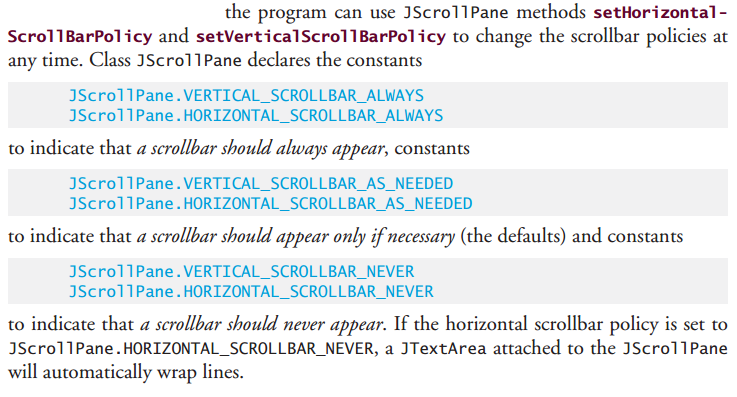
****

****

* **To adjust Layout to postion:**
* **layout.setAlignment( FlowLayout.LEFT);**
* **if we use an action to change layout so we can realign it:**
* **layout.layoutContainer( container );**

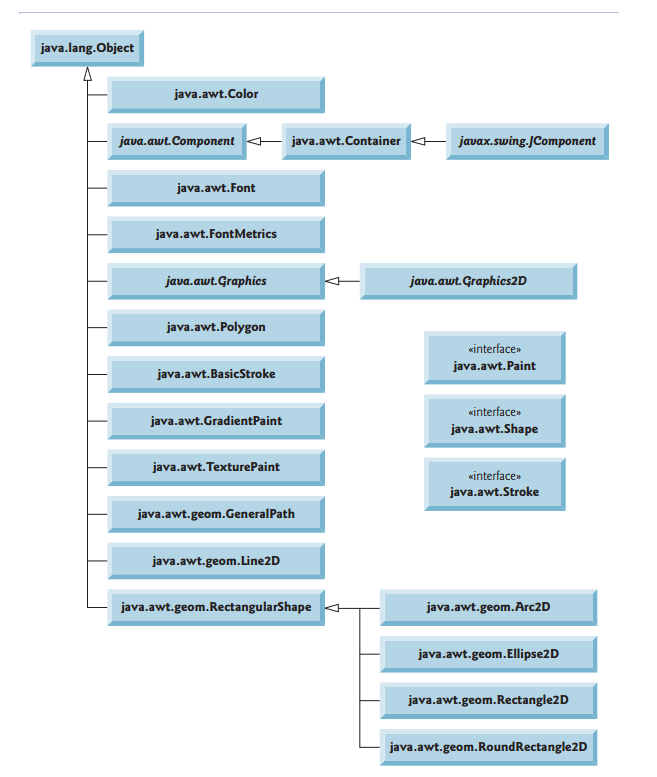
**// to specify that the JFrame should be rearranged based on the adjusted layout.**

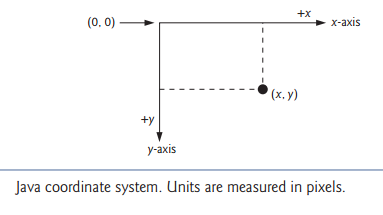
* **Each frame contains three containers as the back, glass and the one that contain component so we can hold each of them in container**
* **Container c = FrmName.getContentpane();**
* **A BorderLayout limits a Container to containing at most five components—one in each region. The component placed in each region can be a container to which other components are attached.**
* **The constructor arguments specify the number of pixels between components that are arranged horizontally (horizontal gap space) and between components that are arranged vertically (vertical gap space), respectively. The default isone pixel of gap space horizontally and vertically.**
* **If no region is specified when adding a Component to a BorderLayout, the layout manager assumes that the Component should be added to region BorderLayout.CENTER.**
* **When more than one component is added to a region in a BorderLayout, only the last component added to that region will be displayed. There’s no error that indicates this problem.**
* **BorderLayout l = new Borderlayout(5,5);**
* **f.add(B1,BorderLayout.WEST);**
* **The GridLayout layout manager divides the container into a grid so that components can be placed in rows and columns; Every Component in a GridLayout has the same width and height. Components are added to a GridLayout starting at the top-left cell of the grid and proceeding left to right until the row is full. Then the process continues left to right on the next row of the grid, and so on.**
* **gridLayout1 = new GridLayout( 2, 3,5,5); //2by3;gapsof5**
* **gridLayout2 = new GridLayout( 3, 2 );//3by2;nogaps**
* **to change the grid layout for the frome we use**
* **FrmName.getContentpane().setLayout(GridLayName);**
* **container.validate(); // re-lay out container**
* **Class JPanel extends JComponent and JComponent extends class Container, so every JPanel is a Container. Thus, every JPanel may have components, including other panels, attached to it with Container method add**
* **To get selected text in textArea:**
* **textArea1.getSelectedText()**
* **Box’s static method createHorizontalBox creates a Box that arranges components from left to right in the order that they’re attached.**
* **To add scroll bar to the textArea:**
* **FrmName.add(new JScrollPane(JTextName));**
* **To provide line wrapping functionality for a JTextArea, invoke JTextArea method setLineWrap with a true argument**
* **jt.setLineWrap(true);**

****

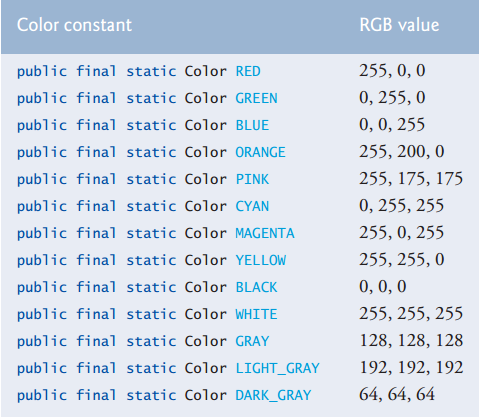
**Like:**

* **js.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL\_SCROLLBAR\_NEVER);**
* **js.setVerticalScrollBarPolicy(JScrollPane.VERTICAL\_SCROLLBAR\_NEVER);**
* **Coordinate units are measured in pixels (which stands for “picture element”). A pixel is a display monitor’s smallest unit of resolution.**
* **Different display monitors have different resolutions (i.e., the density of the pixels varies). This can cause graphics to appear in different sizes on different monitors or on the same monitor with different settings.**

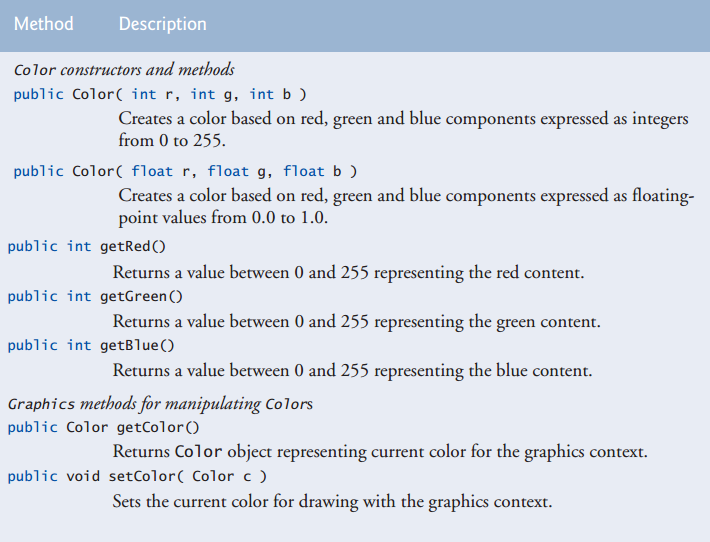
****

****

* **Class Graphics is an abstract class. This contributes to Java’s portability. Because drawing is performed differently on every platform that supports Java, there cannot be only one implementation of the drawing capabilities across all systems. For example, the graphics capabilities that enable a PC running Microsoft Windows to draw a rectangle are different from those that enable a Linux workstation to draw a rectangle**
* **Class Component is the superclass for many of the classes contains a paintComponent method that can be used to draw graphics. Method paintComponent takes a Graphics object as an argument. This object is passed to the paintComponent method by the system when a lightweight Swing component needs to be repainted.**
* **Java uses a multithreaded model of program execution. Each thread is a parallel activity. Each program can have many threads. When you create a GUI-based application, one of those threads is known as the event-dispatch thread (EDT)—it’s used to process all GUI events. All drawing and manipulation of GUI components should be performed in that thread.**
* **For paintComponent to be called again, an event must occur (such as covering and uncovering the component with another window). If you need paintComponent to execute you can call method repaint.**
* **Predeclared colors in Java**

****

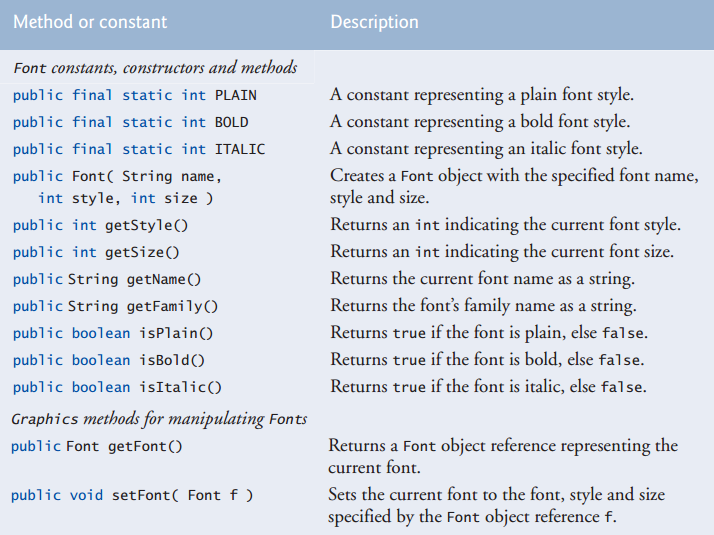
* **Color methods**

****

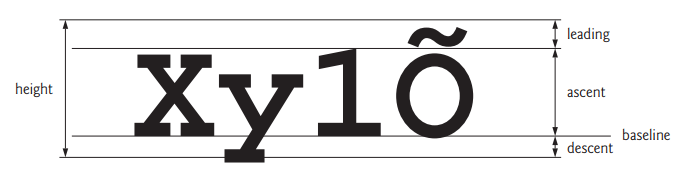
* **All three RGB components can be integers in the range from 0 to 255, or all three can be floating-point values in the range 0.0 to 1.0. The first RGB component specifies the amount of red, the second the amount of green and the third the amount of blue.**
* **Some Graphics codes:**
* **g.drawString("Current RGB: "+,130,40);**
* **Color color = Color.MAGENTA;**
* **color = JColorChooser.showDialog( ShowColors2JFrame.this,"Choose a color", color );**

**This method is used to let the user choose the color of some thing**

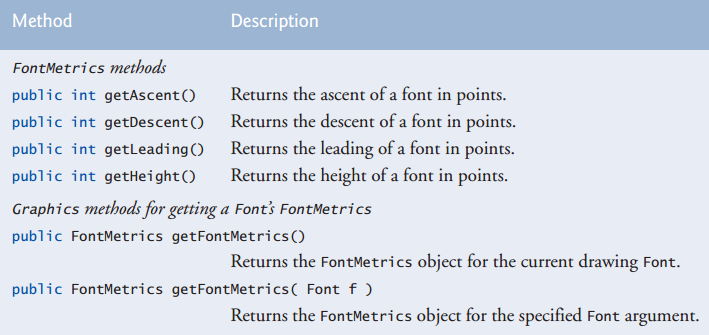
* **The method takes three arguments—a reference to its parent Component, a String to display in the title bar of the dialog and the initial selected Color for the dialog**
* **The parent component is a reference to the window from which the dialog is displayed (in this case the JFrame, with the reference name frame). The dialog will be centered on the parent. If the parent is null, the dialog is centered on the screen.**
* **The HSB tab allows you to select a color based on hue, saturation and brightness—values that are used to define the amount of light in a color. We do not discuss HSB values. For more information on them**
* **Font methods**

****

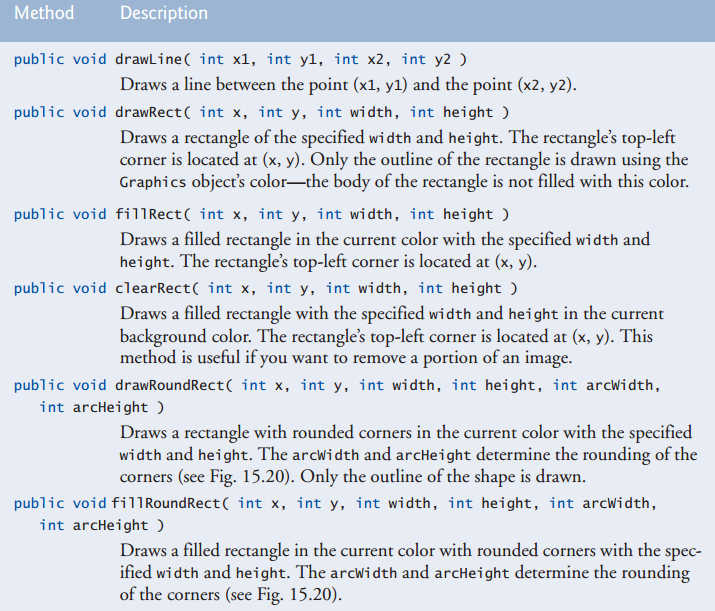
* **The number of fonts varies across systems. Java provides five font names—Serif, Monospaced, SansSerif, Dialog and DialogInput—that can be used on all Java platforms. The Java runtime environment (JRE) on each platform maps these logical font names to actual fonts installed on the platform. The actual fonts used may vary by platform.**
* **To change the font, you must create a new Font object. Font objects are immutable—class Font has no setmethods to change the characteristics of the current font.**
* **Once Graphics method setFont is invoked, all text displayed following the call will appear in the new font until the font is changed. Each font’s information is displayed in using method drawstring**
* **Font Matrices:**

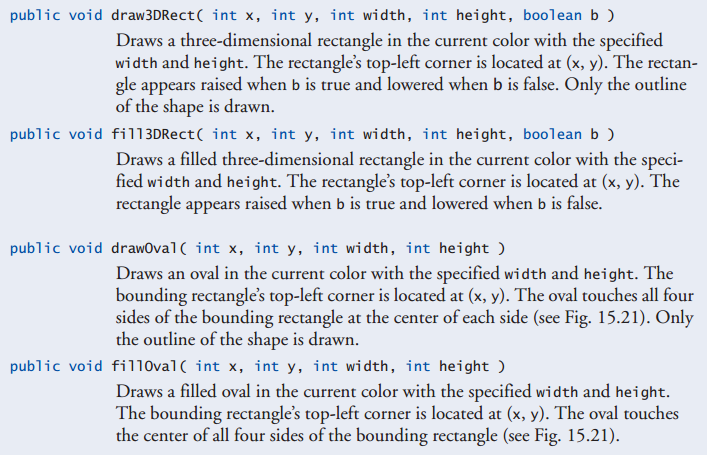
****

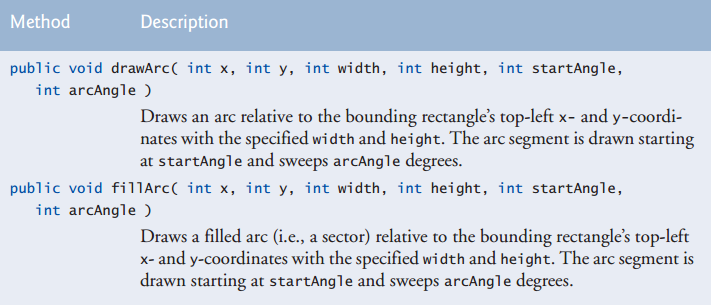
* **Font Matrices and its methods:**

****

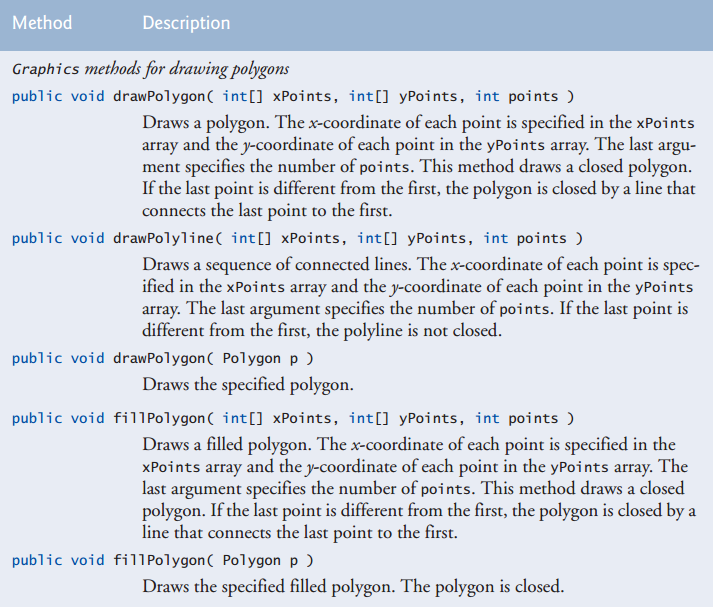
* **Some of its codes:**
* **FontMetrics metrics = g.getFontMetrics();**
* **metrics = g.getFontMetrics( font );**
* **graphics methods:**

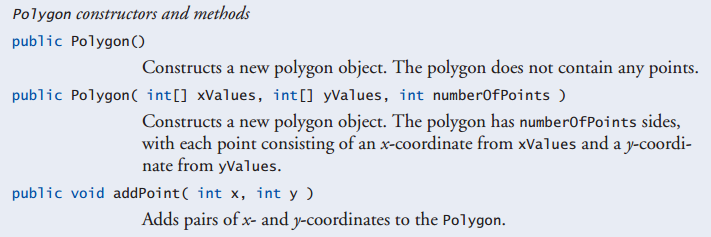
****

****

****

* **polygon methods:**

****

****

* **some good methods for polygon:**
* **Polygon polygon2 = newPolygon();**
* **polygon2.addPoint( 165, 135);**
* **An ArrayIndexOutOfBounds Exception is thrown if the number of points specified in the third argument to method drawPolygon or method fillPolygon is greater than the number of elements in the arrays of coordinates that specify the polygon to display.**
* **Class GradientPaint helps draw a shape in gradually changing colors—called a gradient. The GradientPaint constructor used here requires seven arguments. The first two specify the starting coordinate for the gradient. The third specifies the starting Color for the gradient. The fourth and fifth specify the ending coordinate for the gradient. The sixth specifies the ending Color for the gradient. The last argument specifies whether the gradient is cyclic (true) or acyclic (false).**
* **Some methods from Graphics2D class:**
* **Graphics2D g2d = ( Graphics2D ) g; // cast g to Graphics2D**
* **g2d.setPaint( new GradientPaint ( 5,30,Color.BLUE,35, 100, Color.YELLOW, true ));**
* **g2d.fill(new Ellipse2D.Double(5,30,65, 100));**
* **g2d.setPaint( Color.RED );**
* **g2d.draw(new Rectangle2D.Double( 80, 30, 65, 100));**
* **g2d.draw( newArc2D.Double(240,30,75, 100,0, 270, Arc2D.PIE));**
* **g2d.setStroke( newBasicStroke(6.0f));**
* **gg.fillRect (0, 0, 10,10); // draw a filled rectangle**
* **gg.setColor (Color.BLACK ); // draw in black**
* **gg.drawRect (1, 1, 6,6 );// draw a rectangle**
* **Ellipse2D.Float and Line2D.Float. In each case, Double is a public static nested class of the class specified to the left of the dot (e.g.,Ellipse2D). To use the static nested class, we simply qualify its name with the outer class name.**
* **Method setStroke requires as its argument an object that implements interface Stroke. In this case, we use an instance of class BasicStroke. Class BasicStroke provides several constructors to specify the width of the line, how the line ends (called the end caps), how lines join together (called line joins) and the dash attributes of the line (if it’s a dashed line)**
* **ConstantArc2D.PIEindicates that the**
* **Arc is closed by drawing two lines—one line from the arc’s starting point to the center of the bounding rectangle and one line from the center of the bounding rectangle to the ending point. Class Arc2Dprovides two other static constants for specifying how the arc is closed. ConstantArc2D.CHORD draws a line from the starting point to the ending point. ConstantArc2D.OPEN specifies that the arc should not be closed.**
* **To draw dashed line:**
* **gg.setStroke( new BasicStroke(2,BasicStroke.CAP\_ROUND,BasicStroke.JOIN\_ROUND, 10 , new float[]{3},0));**
* **To draw general path 2d graphics:**
* **GeneralPath star = newGeneralPath(); // create GeneralPath object**

1. **// set the initial coordinate of the General Path**

**star.moveTo( xPoints[ 0], yPoints[ 0 ]);**

1. **star.lineTo( xPoints[ count ], yPoints[ count ] ) // this creat the shape not draw it**
2. **star.closePath(); // close the shape**

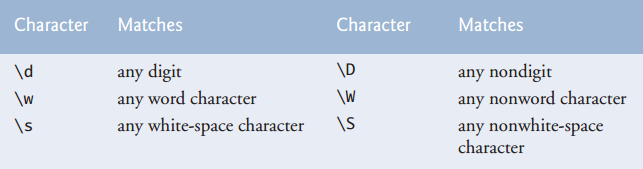
* **g2d.translate( 150, 150); // translate the origin to (150, 150)**
* **g2d.rotate( Math.PI/ 10.0 ); // rotate coordinate system**
* **g2d.fill( star ); // draw filled star –the path specified-**
* **Use General Path method closePath to draw a line from the last point tothe point specified in the last call to moveTo.**
* **A character literal is an integer value represented as a character in single quotes. For example, 'z' represents the integer value of z**
* **The value of a character literal is the integer value of the character in the Unicode character set**
* **To conserve memory, Java treats all string literals with the same contents as a single String object that has many references to it.**
* **String s4 = new String( charArray,6, 3);**

**Define new string from char array from letter 6 and take 3 letters only**

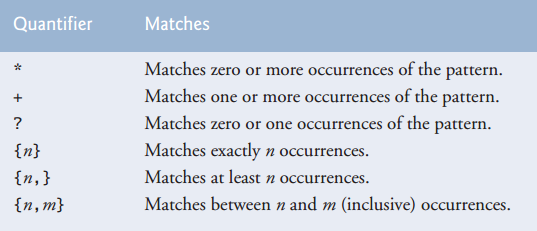
* **Comparing references with==can lead to logic errors, because==compares the references to determine whether they refer to the same object, not whether two objects have the same contents. When two identical (but separate) objects are compared with ==, the result will be false. When comparing objects to determine whether they have the same contents, use method equals**
* **Method equalsIgnoreCase, which ignores whether the letters in each String are uppercase or lowercase when performing the comparison.**
* **Method regionMatches to compare portions of two Strings for equality. The first argument is the starting index in the String that invokes the method. The second argument is a comparison String. The third argument is the starting index in the comparison String. The last argument is the number of characters to compare between the two Strings.**
* **S.startsWith(“good”); //returns true if it starts with good**
* **S.endsWith(“good”); //returns true if it ends with good**
* **S = S.replace(‘l’,’d’); //replace each l character with d character**
* **S = String.valueOf(10); //to convert any object to string**
* **S = String.ValueOf(charArray , 3 , 3); //convert part of char array start at index 3 and take three characters**
* **To add right click menu to a component:**
* **JPopupMenu jp = new JPopupMenu();**
* **jp.add(new JMenuItem("space"));**
* **jp.add(new JMenuItem("space2"));**
* **text.setComponentPopupMenu(jp);**
* **StringBuilders are not thread safe. If multiple threads require access to the same dynamic string information, use class StringBuffer in your code. Classes StringBuilder and StringBuffer provide identical capabilities, but class StringBuffer is thread safe.**
* **StringBuilder constructors**
* **the no-argument constructor to create a StringBuilder with no characters in it and an initial capacity of 16 characters (the default for a StringBuilder).**
* **The StringBuilder constructor that takes an integer argument to create a StringBuilder with no characters in it and the initial capacity specified by the integer argument (i.e.,10).**
* **The StringBuilder constructor that takes a String argument to create a StringBuilder containing the characters in the String argument. The initial capacity is the number of characters in the String argument plus 16.**
* **Method length returns the number of characters currently in a StringBuilder**
* **Method capacity returns the number of characters that can be stored in a StringBuilder without allocating more memory.**
* **Method ensureCapacity guarantees that a StringBuilder has at least the specified capacity.**
* **Method setLength increases or decreases the length of a StringBuilder**
* **Dynamically increasing the capacity of a StringBuilder can take a relatively long time. Executing a large number of these operation scan degrade the performance of an application. If a StringBuilder is going to increase greatly in size, possibly multiple times, setting its capacity high at the beginning will increase performance.**
* **To copy some chars from StringBuilder to char array:**
* **x.getChars(0, 3, charArray, 2);**
* **first and second argument are start and end index in StringBuilder**
* **third argument is the char array we copy data to it**
* **the forth argument is the start index in the char array to copy to it**
* **to add content to StringBuilder we have two ways:**
* **s += "!";**
* **s = new StringBuilder().append( s ).append( "!" ).toString();**
* **some other methods for StringBuilder:**
* **buffer.insert( 0, objectRef );**
* **buffer.deleteCharAt( 10); // delete 5 in 2.5**
* **buffer.delete( 2,6 );// delete .333 in 33.333**
* **The wrapper class Character contains some methods such as:**
* **Character.isDefined(c); //** **Character method isDefined to determine whether character c is defined in the Unicode character set. If so, the method returns true; otherwise, it returns false**
* **Character.isDigit(c);**
* **Character.isDigit(10); //** **method isDigit to determine whether character c is a defined Unicode digit. If so, the method returns true, and otherwise, false**
* **Character.isJavaIdentifierStart(c); //** **to determine whether c is**

**a character that can be the first character of an identifier in Java—that is, a letter, an underscore (\_) or a dollar sign ($). If so, the method returns true, and otherwise, false**

* **Character.isJavaIdentifierPart(c); //** **to determine whether character c is a character that can be used in an identifier in Java—that is, a digit, a letter, an underscore (\_) or a dollar sign ($). If so, the method returns true, and otherwise, false.**
* **Character.isLetter(c);**
* **Character.isLetterOrDigit(c);**
* **Character.isLowerCase(c);**
* **Character.isUpperCase(c);**
* **Character.toLowerCase(c);**
* **Character.toUpperCase(c);**
* **Character.forDigit(40, c);**
* **Character.digit(c, 10);**
* **new Character(‘c’).charValue( );**
* **A regular expression is a String that describes a search pattern for matching characters in other Strings.**
* **Method matches receives a String that specifies the regular expression and matches the contents of the String object on which it’s called to the regular expression. The method returns a boolean indicating whether the match succeeded.**
* **A regular expression consists of literal characters and special symbols.**
* **A character class is an escape sequence that represents a group of characters. A digit is any numeric character.**
* **A word character is any letter (uppercase or lowercase), any digit or the underscore character.**
* **A white-space character is a space, a tab, a carriage return, a newline or a form feed.**
* **Each character class matches a single character in the String we’re attempting to match with the regular expression.**

****

* **To match a set of characters that does not have a predefined character class, use square brackets, []. For example, the pattern"[aeiou]"matches a single character that’s a vowel.**
* **Character ranges are represented by placing a dash (-) between two characters. In the example, "[A-Z]" matches a single uppercase letter.**
* **If the first character in the brackets is"^", the expression accepts any character other than those indicated. Like "[^Z]"**
* **The range"[A-z]"matches all letters and also matches those characters (such as [ and \) with an integer value between uppercase Z and lowercase a**
* **The asterisk after the second character class indicates that any number of letters can be matched. In general, when the regular-expression operator"\*"appears in a regular expression, the application attempts to match zero or more occurrences of the sub expression**
* **So both "A\*" and "A+" will match "AAA" or "A" ,but only "A\*" will match an empty string**
* **The character "|" matches the expression to its left or to its right. For example, "Hi ( John | Jane)"matches both "Hi John" and "Hi Jane"**
* **The parentheses are used to group parts of the regular expression.**
* **For example for String s = “Mahmoud”;**
* **s.matches("M[a-w]\*");**
* **city.matches("([a-zA-Z]+ | [a-zA-Z]+\\s[a-zA-Z]+)");**
* **zip.matches("\\d{5}")**
* **s.matches("//d{2}")**
* **All quantifiers affect only the sub expression immediately preceding the quantifier**
* **All of the quantifiers are greedy. This means that they’ll match as many occurrences as they can as long as the match is still successful**
* **If any qualifier is** **followed by a question mark (?), the quantifier becomes reluctant (sometimes called lazy). It then will match as few occurrences as possible as long as the match is still successful.**
* **String method matches checks whether an entire String conforms to a regular expression. For example, we want to accept "Smith" as a last name, but not"9@Smith#". If only a substring matches the regular expression, method matches returns false.**

****

* **The regular expression ("\\\*") precedes character\*with two backslashes. Normally,\*is a quantifier indicating that a regular expression should match any number of occurrences of a preceding pattern**
* **CharSequence is an interface that allows read access to a sequence of characters. The interface requires that the methods charAt, length, subsequence and toString be declared. Both String and StringBuilder implement interface CharSequence, so an instance of either of these classes can be used with class Matcher.**
* **A regular expression can be tested against an object of any class that implements interface CharSequence, but the regular expression must be a String. Attempting to create a regular expression as a StringBuilder is an error**
* **If a regular expression will be used only once, static Pattern method matches can be used.**
* **If a regular expression will be used more than once (in a loop, for example), it’s more efficient to use static Pattern method compile to create a specific Pattern object for that regular expression. This method receives a String representing the pattern and returns a new Pattern object**
* **To use Pattern and Matcher classes:**
* **Pattern expression = Pattern.compile("J.\*\\d[0-35-9]-\\d\\d-\\d\\d");**

**// J.\* means all characters after J if the condition is true**

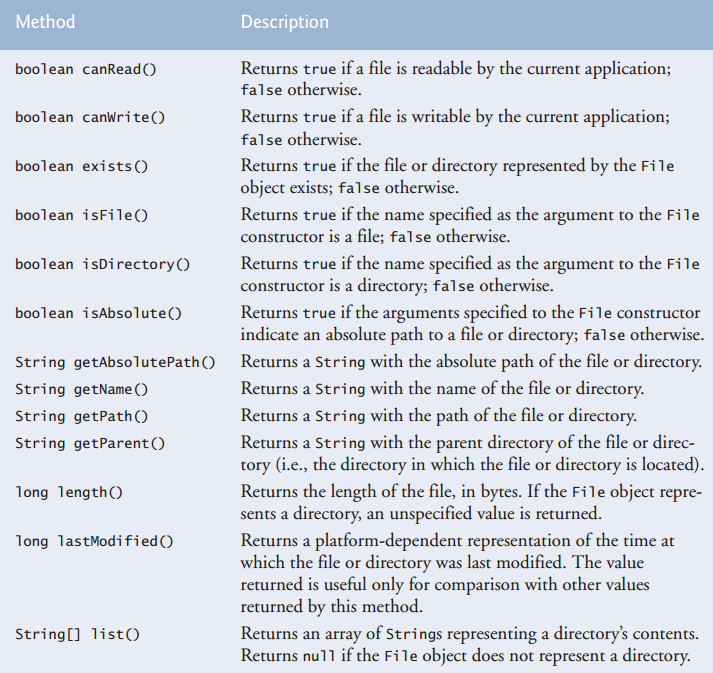
**//** **the dot character "." in the regular expression matches any single character except a newline character.**

* **Matcher matcher = expression.matcher( string1 );**
* **while( matcher.find() )**
* **System.out.println( matcher.group() );**
* **Pattern.compile("\\d[2-3 5-8]"); //specifies two digit ranges that the number will be through it**
* **Matcher method find used to attempt to match a piece of the search object to the search pattern. Each call to this method starts at the point where the last call ended, so multiple matches can be found**
* **Matcher method lookingAt performs the same way, except that it always starts from the beginning of the search object and will always find the first match if there is one.**
* **Method matches(from class String, Pattern or Matcher) will return true only if the entire search object matches the regular expression. Methods find and lookingAt (from class Matcher) will return true if a portion of the search object matches the regular expression.**
* **Matcher method group, which returns the String from the search object that matches the search pattern. The String that’s returned is the one that was last matched by a call to find or lookingAt**

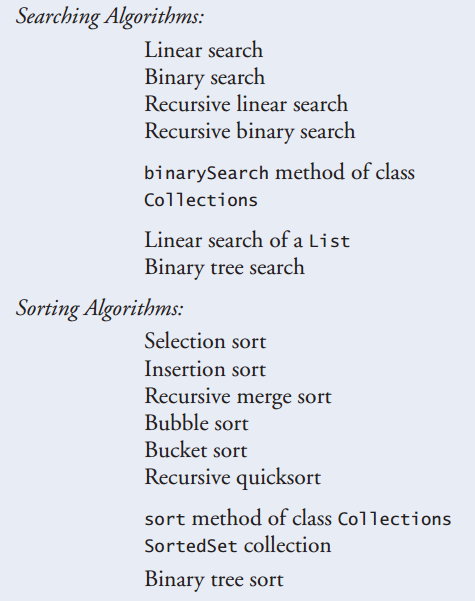
**Files:**

* **Each stream can be redirected. For System.in, this capability enables the program to read bytes from a different source. For System.out and System.err, it enables the output to be sent to a different location, such as a file on disk. Class System provides methods setIn, setOut and setErr to redirect the standard input, output and error streams, respectively.**
* **A Uniform Resource Identifier (URI)is a more general form of the Uniform Resource Locators (URLs)that are used to locate websites. For example, http://www.deitel.com/is the URL for the Deitel & Associates website. URIs for locating files varies across operating systems.**

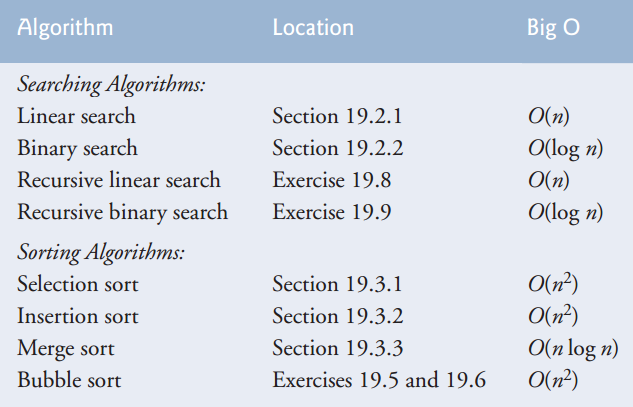
**Like:** [**file://study:/My study**](file://study:/My%20study)

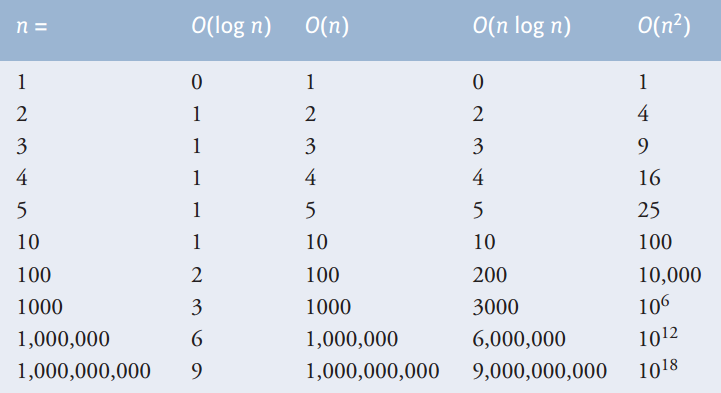
****

* **Using\as a directory separator rather than\\in a string literal is a logic error. A single \indicates that the\followed by the next character represents an escape sequence. Use \\ to insert a\in a string literal.**
* **Method System.exit and pass the value1. This method terminates the application. An argument of0to method exit indicates successful program termination. A nonzero value, such as 1in this example, normally indicates that an error has occurred. This value is passed to the command window that executed the program. The argument is useful if the program is executed from a batch file on Windows systems or a shell script on UNIX/ Linux/Mac OS X systems. Batch files and shell scripts offer a convenient way of executing several programs in sequence. When the first program ends, the next program begins execution. It’s possible to use the argument to method exit in a batch file or shell script to determine whether other programs should execute.**
* **If you use a text editor to open the clients. Txt file produced, each record might not display on a separate line. For example, in Notepad (Microsoft Windows), users will see one continuous line of text. This occurs because different platforms use different line-separator characters. On UNIX/ Linux/Mac OS X, the line separator is a newline (\n). On Windows, it’s a combination of a carriage return and a linefeed—represented as\r\n. You can use the %n format specifier in a format control string to output a platform-specific line separator, thus ensuring that the text file can be opened and viewed correctly in a text editor for the platform on which the file was created. Also, regardless of the line separator used in a text file, a Java program can still recognize the lines of text and read them.**
* **To use scanner to read from a file:**
* **Scanner s = new Scanner(new File("work.txt"));**
* **System.out.println(s.nextLine());**
* **To create new file in operating system using code:**
* **File f = new File("wrdk.txt");**
* **f.createNewFile();**
* **Or you can delete it:**
* **f.delete();**
* **Sometimes we’ll not know exactly how the data is stored in a file. In such cases, we want to read or write an entire object from a file. Java provides such a mechanism, called object serialization. A so-called serialized object is an object represented as a sequence of bytes that includes the object’s data as well as information about the object’s type and the types of data stored in the object. After a serialized object has been written into a file, it can be read from the file and deserialized —that is, the type information and bytes that represent the object and its data can be used to recreate the object in memory.**
* **Classes ObjectInputStream and ObjectOutputStream, which respectively implement the ObjectInput and ObjectOutput interfaces, enable entire objects to be read from or written to a stream (possibly a file). To use serialization with files, we initialize ObjectInputStream and ObjectOutputStream objects with stream objects that read from and write to files**
* **Initializingstreamobjectswithotherstreamobjectsinthismannerissometimescalled wrapping—the new stream object being created wraps the stream object specified as a constructor argument.**
* **Interface Serializable is a tagging interface. Such an interface does not contain methods, This is important, because an ObjectOutputStream will not output an object unless it is a Serializable object**
* **In a Serializable class, every instance variable must be Serializable. Non-Serializable instance variables must be declared transient to indicate that they should be ignored during the serialization process. By default, all primitive-type variables are serializable. For reference-type variables, you must check the class’s documentation (and possibly its superclasses) to ensure that the type is Serializable. For example, Strings are Serializable. By default, arrays are serializable; however, in a reference-type array, the referenced objects might not be.**
* **It’s a logic error to open an existing file for output when, in fact, you wish to preserve the file. Class FileOutputStream provides an overloaded constructor that enables you to open a file and append data to the end of the file. This will preserve the file’s contents.**
* **To save objects with serialization:**
* **ObjectOutputStream output = new ObjectOutputStream( newFileOutputStream("clients.ser"));**
* **Ouput.writeObject(yourObject);**
* **Output.close();**
* **To read an object from a serialized file:**
* **ObjectInputStream input = new ObjectInputStream( new FileInputStream("work.txt"));**
* **yourObject = (ObjectClass) input.read();**
* **input.close();**
* **The call to method close is contained in a try block. Method close throws an IOException if the file cannot be closed properly. In this case, it’s important to notify the user that the information in the file might be corrupted. When using wrapped streams, closing the outermost stream also closes the underlying file.**
* **Method readObject throws an EOFException if an attempt is made to read beyond the end of the file.**
* **To open Java file using JFileChosser:**
* **JFileChooser fileChooser = new JFileChooser();**
* **fileChooser.setFileSelectionMode(JFileChooser.FILES\_AND\_DIRECTORIES);**
* **int result = fileChooser.showOpenDialog(this);**
* **if (result == JFileChooser.CANCEL\_OPTION)**
* **File fileName = fileChooser.getSelectedFile(); // get File**
* **You can also choose more than one selection:**
* **fileChooser.setMultiSelectionEnabled(true);**
* **fileChooser.getSelectedFiles();**
* **Recursive problem-solving approaches have a number of elements in common. When a recursive method is called to solve a problem, it actually is capable of solving only the simplest case(s), or base case(s). If the method is called with a base case, it returns a result. If the method is called with a more complex problem, it typically divides the problem into two conceptual pieces—a piece that the method knows how to do and a piece that it does not know how to do**
* **Either omitting the base case or writing the recursion step incorrectly so that it does not converge on the base case can cause a logic error known as infinite recursion, where recursive calls are continuously made until memory is exhausted. This error is analogous to the problem of an infinite loop in an iterative (nonrecursive) solution.**
* **Since BigInteger is not a primitive type, we can’t use the arithmetic, relational and equality operators with BigIntegers; instead, we must use BigInteger methods to perform these tasks. Line 10 tests for the base case using BigInteger method compareTo. This method compares theBigIntegerthat calls the method to the method’s BigInteger argument. The method returns-1if the BigInteger that calls the method is less than the argument,0if they’re equal or 1if the BigInteger that calls the method is greater than the argument.**
* **To work with big integers:**
* **BigInteger i = new BigInteger("3420934832409348329487392489348392482394839480329483094832");**
* **System.out.println(i.multiply(new BigInteger("3420934832409348329487392489348392482394839480329483094832")));**
* **BigInteger.One;**
* **BigInteger.Zero;**
* **Avoid Fibonacci-style recursive programs, because they result in an exponential “explosion” of method calls**
* **Recursion has many negatives. It repeatedly invokes the mechanism, and consequently the overhead, of method calls. This repetition can be expensive in terms of both processor time and memory space. Each recursive call causes another copy of the method (actually, only the method’s variables, stored in the activation record) to be created—this set of copies can consume considerable memory space. Since iteration occurs within a method, repeated method calls and extra memory assignment are avoided.**
* **Any problem that can be solved recursively can also be solved iteratively ( nonrecursively ). A recursive approach is normally preferred over an iterative approach when the recursive approach more naturally mirrors the problem and results in a program that is easier to understand and debug. A recursive approach can often be implemented with fewer lines of code. Another reason to choose a recursive approach is that an iterative one might not be apparent.**
* **Avoid using recursion in situations requiring high performance. Recursive calls take time and consume additional memory.**

****

* **In Arrays class we have:**
* **Arrays.binarySearch();**
* **Arrays.Sort();**
* **To use this methods with an array of your class your class must implement comparable and its method compartTO to make it work**
* **Searching algorithms all accomplish the same goal—finding an element that matches a given search key, if such an element does, in fact, exist. There are, however, a number of things that differentiate search algorithms from one another. The major difference is the amount of effort they require to complete the search. One way to describe this effort is with Big O notation, which indicates the worst-case run time for an algorithm**
* **This results in a big O of O(log n) for a binary search, which is also known as logarithmic run time.**
* **Selection sort is a simple, but inefficient, sorting algorithm. Its first iteration selects the smallest element in the array and swaps it with the first element. The second iteration selects the second-smallest item**
* **The selection sort algorithm runs in O(n2 ) time.**
* **Insertion sort is another simple, but inefficient, sorting algorithm. The first iteration of this algorithm takes the second element in the array and, if it’s less than the first element, swaps it with the first element. The second iteration looks at the third element and inserts it into the correct position with respect to the first two, so all three elements are in order. At the ith iteration of this algorithm, the first i elements in the original array will be sorted**
* **The insertion sort algorithm also runs in O (n2) time. Like selection sort**
* **Merge sort is an efficient sorting algorithm but is conceptually more complex than selection sort and insertion sort. The merge sort algorithm sorts an array by splitting it into two equal-sized sub arrays, sorting each sub array, and then merging them into one larger array. With an odd number of elements, the algorithm creates the two sub arrays such that one has one more element than the other.**

****

****

* **Merge sort code:**
* **We have array of ints called data**

**// calls recursive split method to begin merge sorting**

**public void sort(int[] data) {**

**sortArray(0, data.length - 1 , data);// split entire array**

**}// end method sort**

**//splits array, sorts sub arrays and merges sub arrays into sorted array**

**private void sortArray(int low, int high , int [] data) {**

**// test base case; size of array equals 1**

**if ((high - low) >= 1)// if not base case**

**{**

**int middle1 = (low + high) / 2;// calculate middle of array**

**int middle2 = middle1 + 1;// calculate next element over**

**// split array in half; sort each half (recursive calls)**

**sortArray(low, middle1 , data); // first half of array**

**sortArray(middle2, high , data);// second half of array**

**// merge two sorted arrays after split calls return**

**merge(low, middle1, middle2, high , data);**

**} // end if**

**}// end method sortArray**

**// merge arrays until reaching end of either**

**// merge two sorted subarrays into one sorted subarray**

**private void merge(int left, int middle1, int middle2, int right, int []data) {**

**int leftIndex = left; // index into left subarray**

**int rightIndex = middle2;// index into right subarray**

**int combinedIndex = left;// index into temporary working array**

**int[] combined = new int[data.length];// working array**

**while (leftIndex <= middle1 && rightIndex <= right) {**

**// place smaller of two current elements into result**

**// and move to next space in arrays**

**if (data[ leftIndex] <= data[ rightIndex]) {**

**combined[ combinedIndex++] = data[ leftIndex++];**

**} else {**

**combined[ combinedIndex++] = data[ rightIndex++];**

**}**

**} // end while**

**// if left array is empty**

**if (leftIndex == middle2) // copy in rest of right array**

**{**

**while (rightIndex <= right) {**

**combined[ combinedIndex++] = data[ rightIndex++];**

**}**

**} else // right array is empty**

**// copy in rest of left array**

**{**

**while (leftIndex <= middle1) {**

**combined[ combinedIndex++] = data[ leftIndex++];**

**}**

**}**

**// copy values back into original array**

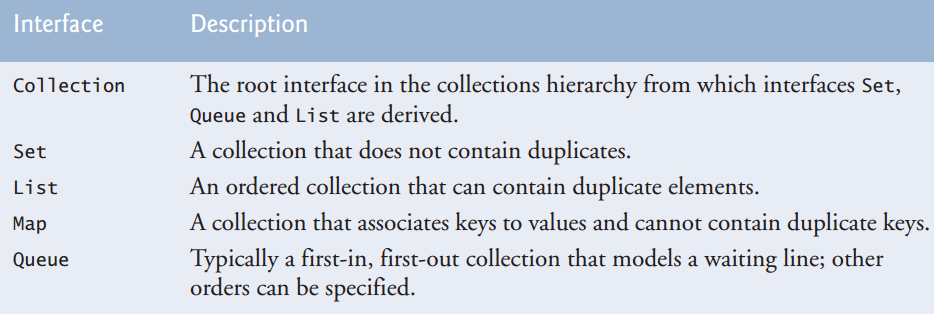
**for (int i = left; i <= right; i++) {**

**data[ i] = combined[ i];**

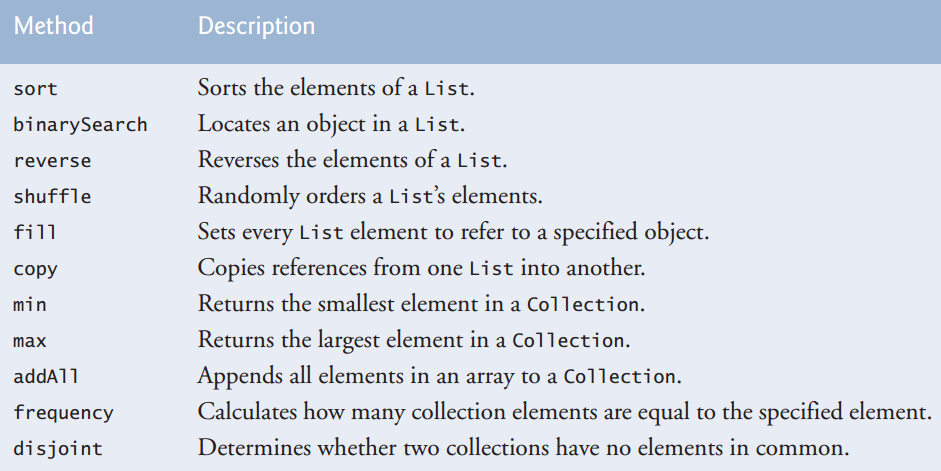
**}**

**}// end method merge**

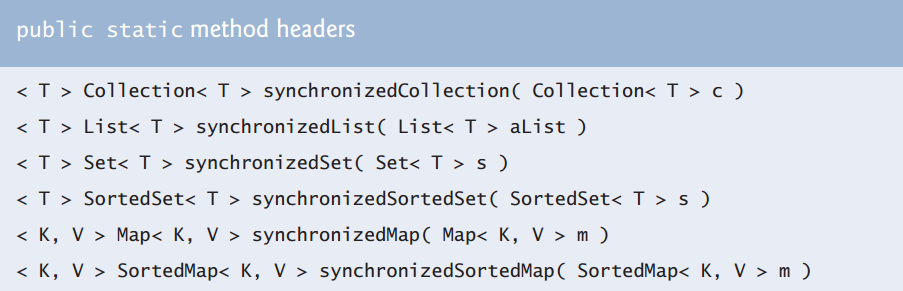
* **Acollectionis a data structure—actually, an object—that can hold references to other objects. Usually, collections contain references to objects that are all of the same type.**

****

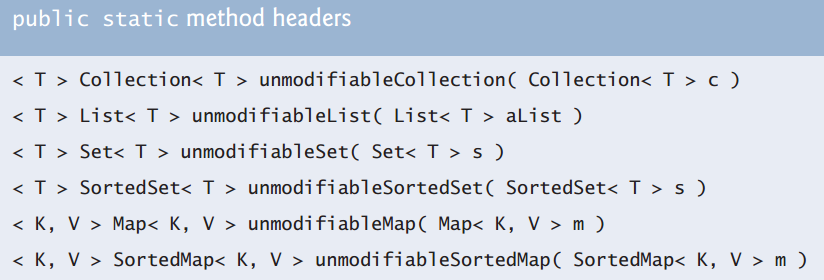
* **Boxing and unboxing -conversion- between wrapper and permitive data type:**
* **Integer[] integerArray = new Integer[5 ]; // create integerArray**
* **integerArray[0]=new Integer(10 );**
* **int value = integerArray[0].intValue();**
* **or**
* **integerArray[0]=10;// assign Integer 10 to integerArray[ 0 ]**
* **int value = integerArray[0]; // get int value of Integer**
* **Collection is used commonly as parameter types in methods to allow polymorphic processing of all objects that implement interface Collection.**
* **We can synchronize access to a collection to ensure that only one thread manipulates the collection at a time. The synchronization wrapper methods of class Collections return synchronized versions of collections that can be shared among threads in a program.**
* **Interface List is implemented by several classes, including ArrayList, LinkedList and Vector**
* **Inserting an element between existing elements of an ArrayList or Vector is an inefficient operation—all elements after the new one must be moved out of the way, which could be an expensive operation in a collection with a large number of elements. A LinkedList enables efficient insertion (or removal) of elements in the middle of a collection**
* **Unsynchronized collections provide better performance than synchronized ones. For this reason, ArrayList is typically preferred over Vector in programs that do not share a collection among threads.**
* **We can specify that a method can take any type of collection like ArrayList or LinkedList:**
* **private static voidremoveColors( Collection< String > x );**
* **Iterator< String > iterator = collection1.iterator();**
* **if( collection2.contains( iterator.next() ) )**
* **iterator.remove(); // remove current Color**
* **If a collection is modified by one of its methods after an iterator is created for that collection, the iterator immediately becomes invalid—operations performed with the iterator after this point throw ConcurrentModificationExceptions. For this reason, iterators are said to be “fail fast”.**
* **Collection has method add all to add all elements of anther collection to it**
* **list2.add( color );**
* **if we don’t need the data in the reference now we can release resources by making this reference point to null**
* **for List we have ListIterator to have more properities:**
* **ListIterator < String > iterator = list.listIterator();**
* **iterator.set( color.toUpperCase() );// convert to upper case**
* **list.subList( start, end ).clear(); // remove items**
* **iterator.hasPrevious();**
* **List method subList to obtain a portion of the List(called a sublist). This is a so-called range-view method, which enables the program to view a portion of the list. The sublist is simply a view into the Liston which subList is called.**
* **Class Arrays provides static method asList to view an array (sometimes called the backing array)asaList collection. A List view allows you to manipulate the array as if it were a list. This is useful for adding the elements in an array to a collection and for sorting array elements.**
* **Arrays.asList( colors )**
* **links.addLast( "red");// add as last item**
* **links.addFirst( "cyan"); // add as first item**
* **Passing an array that contains data to toArray can cause logic errors. If the number of elements in the array is smaller than the number of elements in the list on which toArray is called, a new array is allocated to store the list’s elements—without preserving the array argument’s elements. If the number of elements in the array is greater than the number of elements in the list, the elements of the array (starting at index zero) are overwritten with the list’s elements. Array elements that are not overwritten retain their values.**
* **Class Collections provides several high-performance algorithms for manipulating collection elements as static methods.**



* **The collections framework methods are polymorphic. That is, each can operate on objects that implement specific interfaces, regardless of the underlying implementations.**
* **Collections.sort(x);**
* **Collections.reverse(x);**
* **// copy the contents of list into copyList**
* **Collections.copy( copyList, list );**
* **We can make out class implement comparator interface to let us use the Collections.Sort(x , ourClass.Comparator() ) and sort this collection in the way we want like:**
* **public class TimeComparator implements Comparator<myClass>**
* **{**
* **public int compare(myClass x , myClassy )**
* **In the list:**
* **Collections.sort(x, new GGG(10));**
* **It’s important if we use fill with object to fill collection with it we should use clone with it or we will make all the collection refer to the same object**
* **Collections method binarySearch, which locates an object in a List. If the object is found, its index is returned. If the object is not found, binarySearch returns a negative value –may not be -1 -.**
* **Method binarySearch determines this negative value by first calculating the insertion point and making its sign negative. Then, binarySearch subtracts 1 from the insertion point to obtain the return value, which guarantees that method binarySearch returns positive numbers (>= 0) if and only if the object is found.**
* **Because Stack extends Vector, all public Vector methods can be called on Stack objects, even if the methods do not represent conventional stack operations. For example, Vector method add can be used to insert an element anywhere in a stack—an operation that could “corrupt” the stack. When manipulating a Stack, only methods push and pop should be used to add elements to and remove elements from the Stack, respectively.**
* **Class Number is the superclass of the type-wrapper classes for the primitive numeric types, any integer literal that has the suffix L is a long value. An integer literal without a suffix is an int value. Similarly, any floating-point literal that has the suffix F is a float value. A floating-point literal without a suffix is a double value.**
* **Stack methods:**
* **Pop to get the last element and delete it**
* **Push to add element to it**
* **Peek to get the last element without deleting it**
* **PriorityQueue, which implements the Queue interface, orders elements by their natural ordering as specified by Comparable elements’ compareTo method or by a Comparator object that is supplied to the constructor.**
* **When adding elements to a PriorityQueue, the elements are inserted in priority order such that the highest-priority element (i.e., the largest value) will be the first element removed from the PriorityQueue.**
* **Some priority Queue methods:**
* **Offer 🡪 to add element to the priority Queue in its order location**
* **Class PriorityQueue provides five additional constructors. One of these takes an int and a Comparator object to create a PriorityQueue with the initial capacity specified by the int and the ordering by the Comparator**
* **A Set is an unordered Collection of unique elements (i.e., no duplicate elements). The collections framework contains several Set implementations, including HashSet and TreeSet. HashSet stores its elements in a hash table, and TreeSet stores its elements in a tree.**
* **By definition, Sets do not contain duplicates, so when the HashSet is constructed, it removes any duplicates in the Collection**
* **The collections framework also includes the SortedSet interface (which extends Set)for sets that maintain their elements in sorted order—either the elements’ natural order (e.g., numbers are in ascending order) or an order specified by a Comparator. Class TreeSet implements SortedSet**
* **Set< String > set = new HashSet< String >( values );**
* **SortedSet< String > tree = new TreeSet< String >( Arrays.asList( colors ) );**
* **tree.first() // returns first element**
* **tree.last() // returns last element**
* **tree.headSet(“Orange”); // returns Sorted set of the previous elements**
* **tree.tailSet(“orange”); // returns Sorted set of the next elements**
* **Maps associate keys to values. The keys in a Map must be unique, but the associated values need not be. If a Map contains both unique keys and unique values, it’s said to implement a one-to-one mapping. If only the keys are unique, the Map is said to implement a many to-one mapping—many keys can map to one value.**
* **Three of the several classes that implement interface Map are Hash table, HashMap and TreeMap. Hash tables and Hash Maps store elements in hash tables, and TreeMaps store elements in trees. Interface SortedMap extends Map and maintains its keys in sorted order**
* **Numerous applications have this problem—namely, that either the keys are of the wrong type (e.g., not positive integers that correspond to array subscripts) or they’re of the right type, but sparsely spread over a huge range.**
* **The most popular solution to hash-table collisions is to have each cell of the table be a hash “bucket,” typically a linked list of all the key/value pairs that hash to that cell. This is the solution that Java’s Hashtable and HashMap classes implement. Both Hashtable and HashMap implement the Map interface. The primary differences between them are that HashMap is unsynchronized (multiple threads should not modify a HashMap concurrently) and allows null keys and null values.**
* **A hashtable’s load factor affects the performance of hashing schemes. The load factor is the ratio of the number of occupied cells in the hash table to the total number of cells in the hash table. The closer this ratio gets to 1.0, the greater the chance of collisions.**
* **//first one is the key second one is the value**
* **Map< String, Integer > myMap = new HashMap< String, Integer >();**
* **Set< String > keys = map.keySet(); // get keys**
* **A Properties object is a persistent Hash table that normally stores key/value pairs of strings, by “persistent,” we mean that the Properties object can be written to an output stream (possibly a file) and read back in through an input stream.**
* **Properties table = new Properties(); // create Properties table**
* **// set properties**
* **table.setProperty("color","blue" );**
* **FileOutputStream output = new FileOutputStream( "props.dat");**
* **table.store( output, "Sample Properties"); // save properties**
* **props.load( input ); // load properties**
* **Concurrent access to a Collection by multiple threads could cause indeterminate results or fatal errors. To prevent potential threading problems, synchronization wrappers are used for collections that might be accessed by multiple threads.**
* **A wrapper object receives method calls, adds thread synchronization(to prevent concurrent access to the collection) and delegates the calls to the wrapped collection object.**



* **The Collections class provides a set of static methods that create unmodifiable wrappers for collections. Unmodifiable wrappers throw UnsupportedOperationExceptions if attempts are made to modify the collection.**
* **You can use an unmodifiable wrapper to create a collection that offers read-only access to others, while allowing read/write access to yourself. You do this simply by giving others a reference to the unmodifiable wrapper while retaining for yourself a reference to the original collection.**



* **The collections framework provides various abstract implementations of Collection interfaces from which you can quickly “flesh out” complete customized implementations. These abstract implementations include a thin Collection implementation called an Abstract Collection, a List implementation that allows random access to its elements called an AbstractList, a Map implementation called an AbstractMap, a List implementation that allows sequential access to its elements called an AbstractSequentialList, a Set implementation called an AbstractSet and a Queue implementation called AbstractQueue. To write a custom implementation, you can extend the abstract implementation that best meets your needs, and implement each of the class’s abstract methods. Then, if your collection is to be modifiable, override any concrete methods that prevent modification.**
* **To create new file:**
* **File f = new File("D:\\text.docx");**
* **boolean x = f.createNewFile();**
* **\\ retruns Boolean we have created new file or not**
* **To get the free space in a directory**
* **File f = new File("D:\\");**
* **System.out.println(f.getFreeSpace());**
* **To make file readonly file**
* **f.setReadOnly();**
* **to read content of file:**
* **FileInputStream fileS = new FileInputStream( new File("D:\\text.txt") );**
* **FileOutputStream fileI = new FileOutputStream (new File ( "D:\\texter.txt"));**
* **int c =' ';**
* **while((c = fileS.read()) != -1)**
* **fileI.write(c-10);**
* **}**
* **you may need to use method list(); from class file which return a list of file names in a certain directory.**