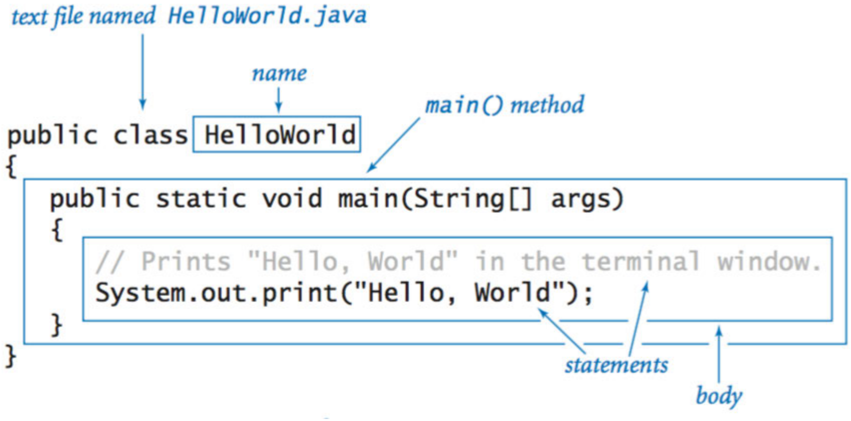
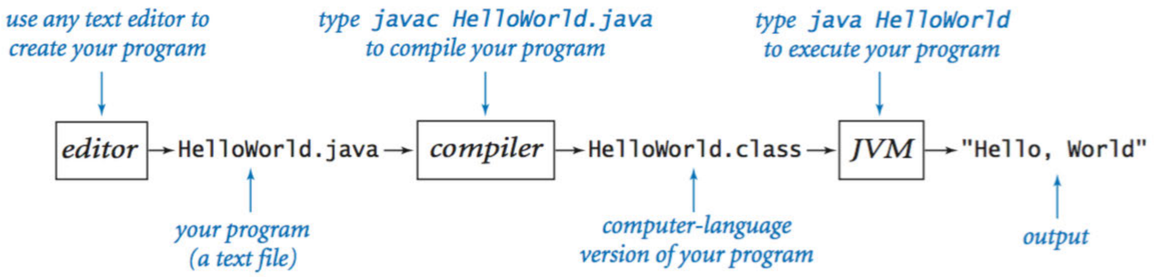
Hello, World.



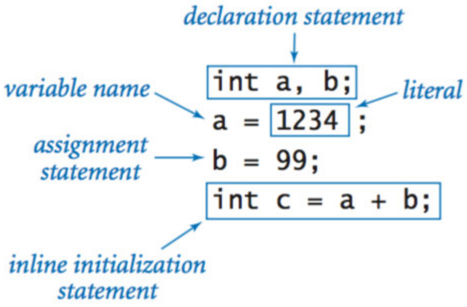
```
public class HelloWorld
{
   public static void main(String[] args)
   {
      // Prints "Hello, World" in the terminal window.
      System.out.print("Hello, World");
   }
}
```

Editing, compiling, and executing.



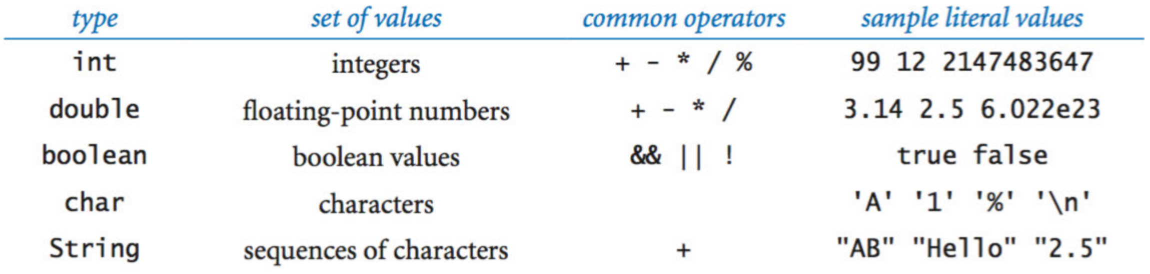
use any text editor to create your program
type javac HelloWorld.java to compile your program
type java HelloWorld to execute your program
editor
HelloWorld.java
compiler
HelloWorld.class
JVM
"Hello, World"
your program (a text file)
computer-language version of your program
output

Declaration and assignment statements.



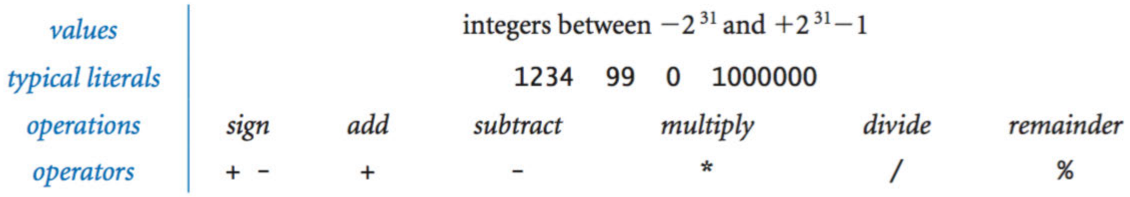
```
int a, b;
a = 1234 ;
b = 99;
int c = a + b;
```

Built-in data types.

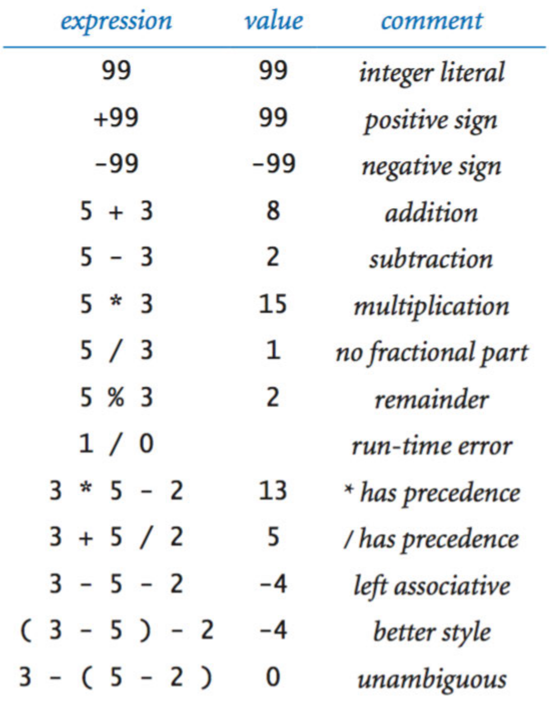


type	set of values	common operators	sample literal values
int	integers	+ - * / %	99 12 2147483647
double	floating-point numbers	+ - * /	3.14 2.5 6.022e23
boolean	boolean values	&& || !	true false
char	characters		'A' '1' '%' '\n'
String	sequences of characters	+	"AB" "Hello" "2.5"

Integers.

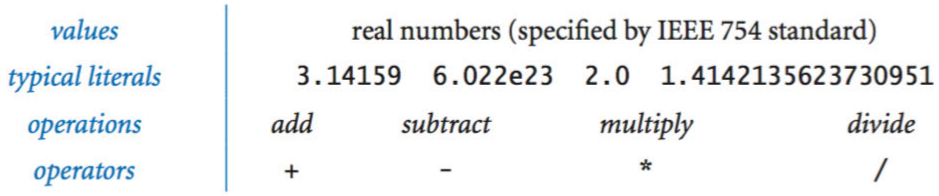


values	integers between -2^{31} and $+2^{31}-1$					
typical literals	1234 99 0 1000000					
operations	sign	add	subtract	multiply	divide	remainder
operators	+ -	+	-	*	/	%

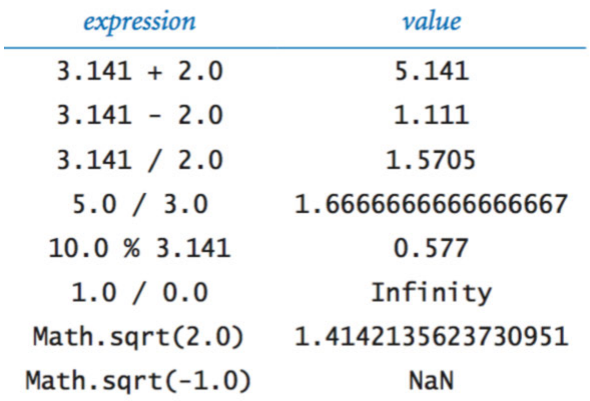


expression	*value*	*comment*
`99`	`99`	*integer literal*
`+99`	`99`	*positive sign*
`-99`	`-99`	*negative sign*
`5 + 3`	`8`	*addition*
`5 - 3`	`2`	*subtraction*
`5 * 3`	`15`	*multiplication*
`5 / 3`	`1`	*no fractional part*
`5 % 3`	`2`	*remainder*
`1 / 0`		*run-time error*
`3 * 5 - 2`	`13`	** has precedence*
`3 + 5 / 2`	`5`	*/ has precedence*
`3 - 5 - 2`	`-4`	*left associative*
`( 3 - 5 ) - 2`	`-4`	*better style*
`3 - ( 5 - 2 )`	`0`	*unambiguous*

Floating-point numbers.

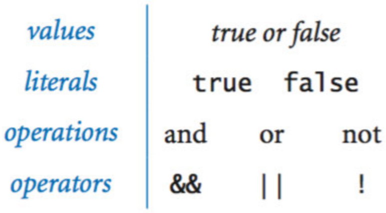


values	real numbers (specified by IEEE 754 standard)			
typical literals	`3.14159`	`6.022e23`	`2.0`	`1.4142135623730951`
operations	*add*	*subtract*	*multiply*	*divide*
operators	`+`	`-`	`*`	`/`

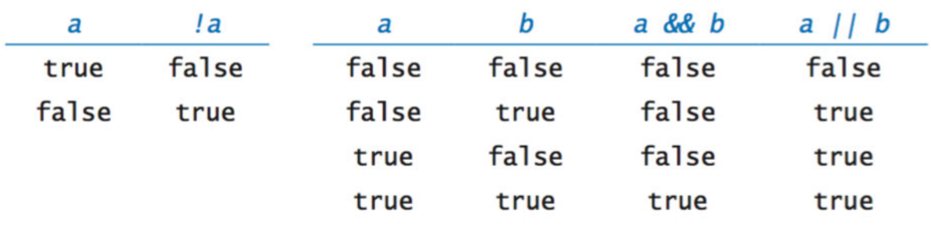


expression	*value*
`3.141 + 2.0`	`5.141`
`3.141 - 2.0`	`1.111`
`3.141 / 2.0`	`1.5705`
`5.0 / 3.0`	`1.6666666666666667`
`10.0 % 3.141`	`0.577`
`1.0 / 0.0`	`Infinity`
`Math.sqrt(2.0)`	`1.4142135623730951`
`Math.sqrt(-1.0)`	`NaN`

Booleans.



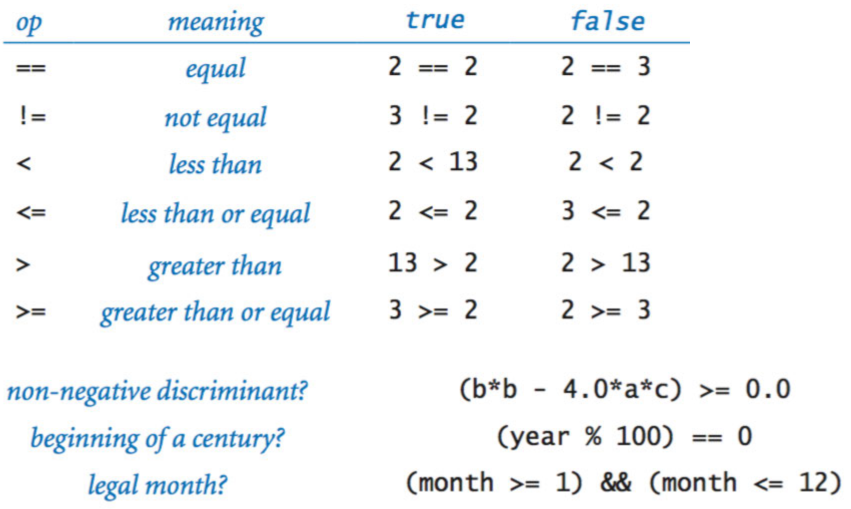
values	*true or false*		
literals	`true`	`false`	
operations	and	or	not
operators	&&	||	!



a	*!a*
`true`	`false`
`false`	`true`

a	*b*	*a && b*	*a || b*
`false`	`false`	`false`	`false`
`false`	`true`	`false`	`true`
`true`	`false`	`false`	`true`
`true`	`true`	`true`	`true`

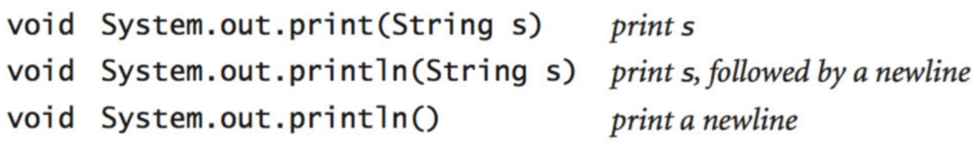
Comparison operators.



op	meaning	true	false
`==`	equal	`2 == 2`	`2 == 3`
`!=`	not equal	`3 != 2`	`2 != 2`
`<`	less than	`2 < 13`	`2 < 2`
`<=`	less than or equal	`2 <= 2`	`3 <= 2`
`>`	greater than	`13 > 2`	`2 > 13`
`>=`	greater than or equal	`3 >= 2`	`2 >= 3`

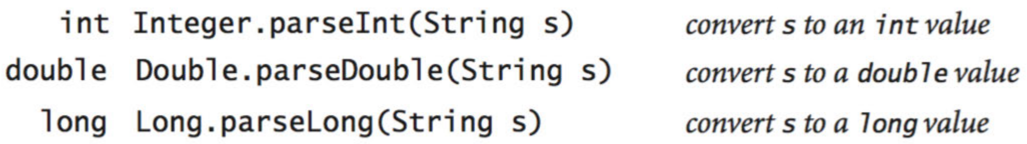
non-negative discriminant?	`(b*b - 4.0*a*c) >= 0.0`
beginning of a century?	`(year % 100) == 0`
legal month?	`(month >= 1) && (month <= 12)`

Printing.



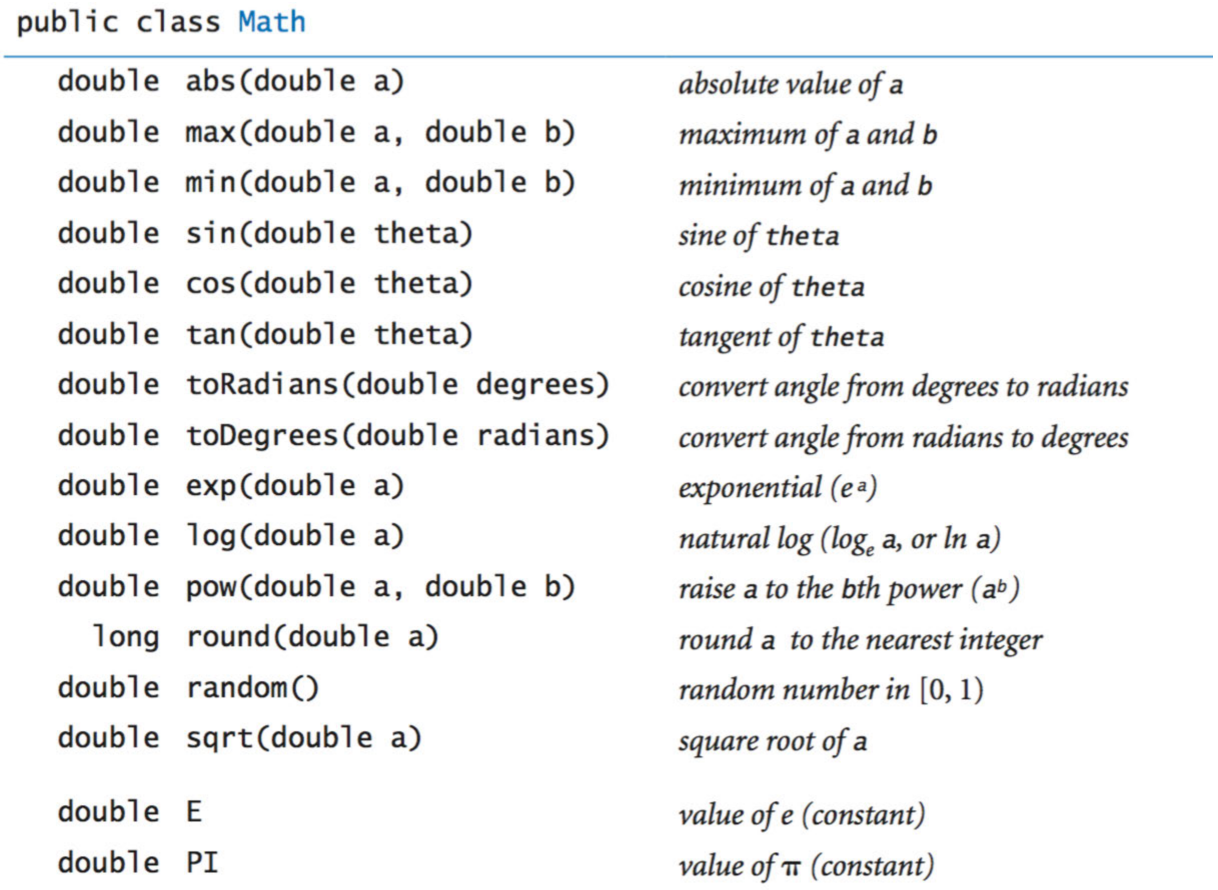
`void System.out.print(String s)`	print s
`void System.out.println(String s)`	print s, followed by a newline
`void System.out.println()`	print a newline

Parsing command-line arguments.



`int Integer.parseInt(String s)`	convert s to an `int` value
`double Double.parseDouble(String s)`	convert s to a `double` value
`long Long.parseLong(String s)`	convert s to a `long` value

Math library.

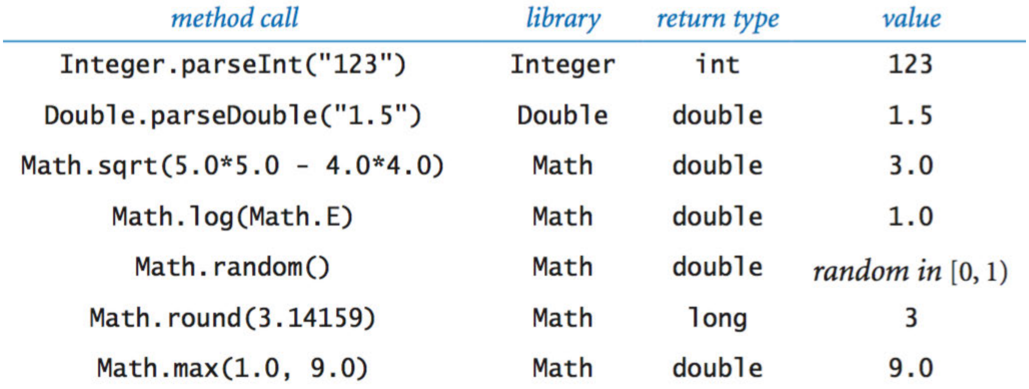


`public class Math`

`double abs(double a)`	absolute value of a
`double max(double a, double b)`	maximum of a and b
`double min(double a, double b)`	minimum of a and b
`double sin(double theta)`	sine of theta
`double cos(double theta)`	cosine of theta
`double tan(double theta)`	tangent of theta
`double toRadians(double degrees)`	convert angle from degrees to radians
`double toDegrees(double radians)`	convert angle from radians to degrees
`double exp(double a)`	exponential (e^a)
`double log(double a)`	natural log ($\log_e$ a, or ln a)
`double pow(double a, double b)`	raise a to the bth power (a^b)
`long round(double a)`	round a to the nearest integer
`double random()`	random number in [0, 1)
`double sqrt(double a)`	square root of a
`double E`	value of e (constant)
`double PI`	value of π (constant)

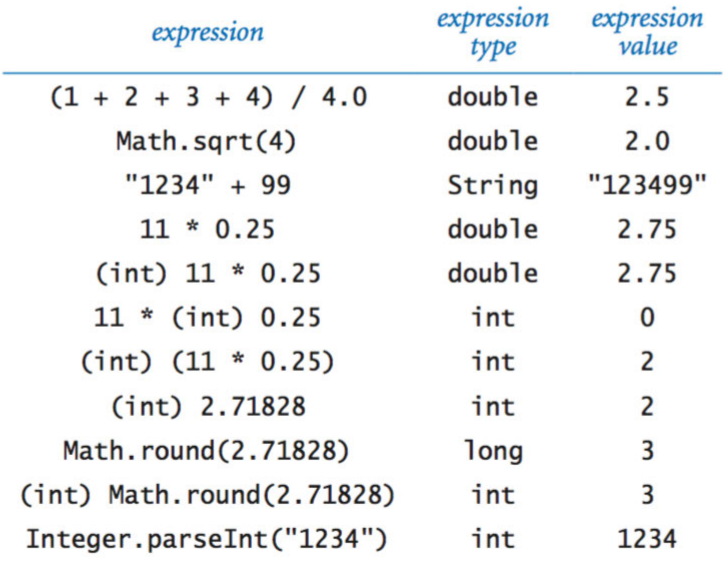
The full java.lang.Math API.

Java library calls.



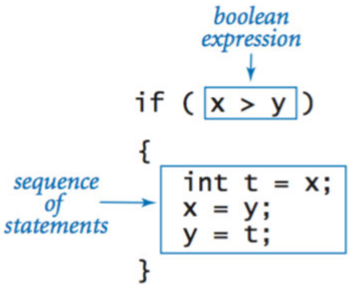
method call	*library*	*return type*	*value*
`Integer.parseInt("123")`	`Integer`	`int`	`123`
`Double.parseDouble("1.5")`	`Double`	`double`	`1.5`
`Math.sqrt(5.0*5.0 - 4.0*4.0)`	`Math`	`double`	`3.0`
`Math.log(Math.E)`	`Math`	`double`	`1.0`
`Math.random()`	`Math`	`double`	*random in* [0, 1)
`Math.round(3.14159)`	`Math`	`long`	`3`
`Math.max(1.0, 9.0)`	`Math`	`double`	`9.0`

Type conversion.



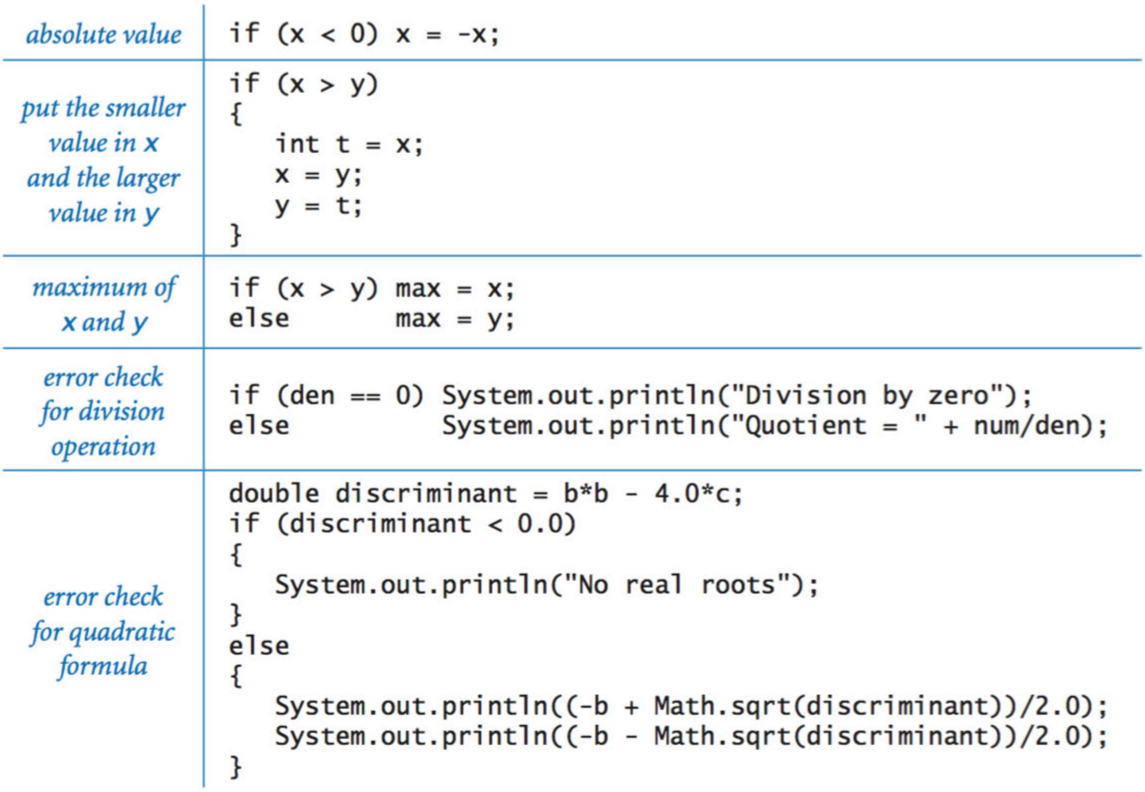
expression	*expression type*	*expression value*
`(1 + 2 + 3 + 4) / 4.0`	`double`	`2.5`
`Math.sqrt(4)`	`double`	`2.0`
`"1234" + 99`	`String`	`"123499"`
`11 * 0.25`	`double`	`2.75`
`(int) 11 * 0.25`	`double`	`2.75`
`11 * (int) 0.25`	`int`	`0`
`(int) (11 * 0.25)`	`int`	`2`
`(int) 2.71828`	`int`	`2`
`Math.round(2.71828)`	`long`	`3`
`(int) Math.round(2.71828)`	`int`	`3`
`Integer.parseInt("1234")`	`int`	`1234`

Anatomy of an if statement.



```
if ( x > y )
{
   int t = x;
   x = y;
   y = t;
}
```

If and if-else statements.



absolute value	`if (x < 0) x = -x;`

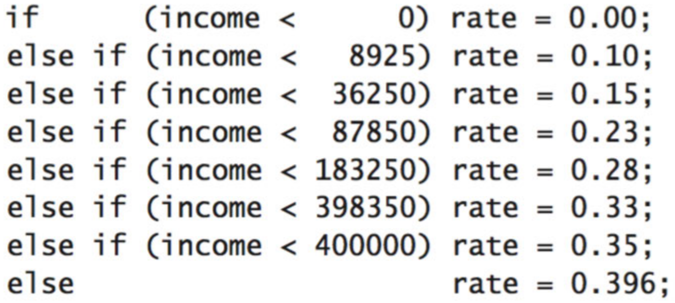
put the smaller value in x and the larger value in y	`if (x > y)`
	`{`
	`   int t = x;`
	`   x = y;`
	`   y = t;`
	`}`

maximum of x and y	`if (x > y) max = x;`
	`else       max = y;`

error check for division operation	`if (den == 0) System.out.println("Division by zero");`
	`else          System.out.println("Quotient = " + num/den);`

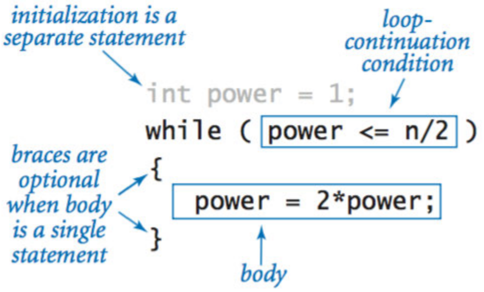
error check for quadratic formula	`double discriminant = b*b - 4.0*c;`
	`if (discriminant < 0.0)`
	`{`
	`   System.out.println("No real roots");`
	`}`
	`else`
	`{`
	`   System.out.println((-b + Math.sqrt(discriminant))/2.0);`
	`   System.out.println((-b - Math.sqrt(discriminant))/2.0);`
	`}`

Nested if-else statement.



```
if      (income <      0) rate = 0.00;
else if (income <   8925) rate = 0.10;
else if (income <  36250) rate = 0.15;
else if (income <  87850) rate = 0.23;
else if (income < 183250) rate = 0.28;
else if (income < 398350) rate = 0.33;
else if (income < 400000) rate = 0.35;
else                      rate = 0.396;
```

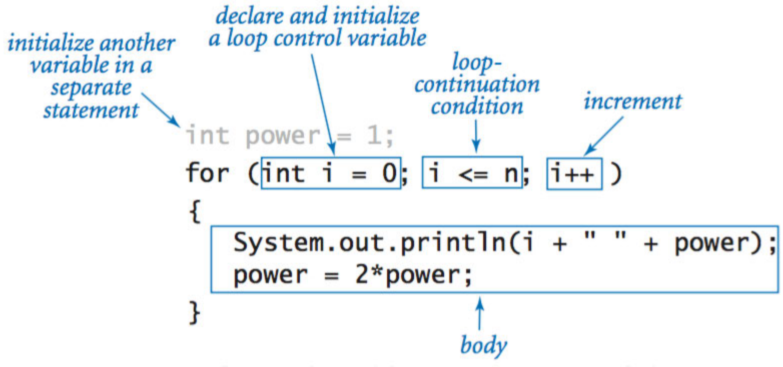
Anatomy of a while loop.



```
int power = 1;
while ( power <= n/2 )
{
   power = 2*power;
}
```

initialization is a separate statement
loop-continuation condition
braces are optional when body is a single statement
body

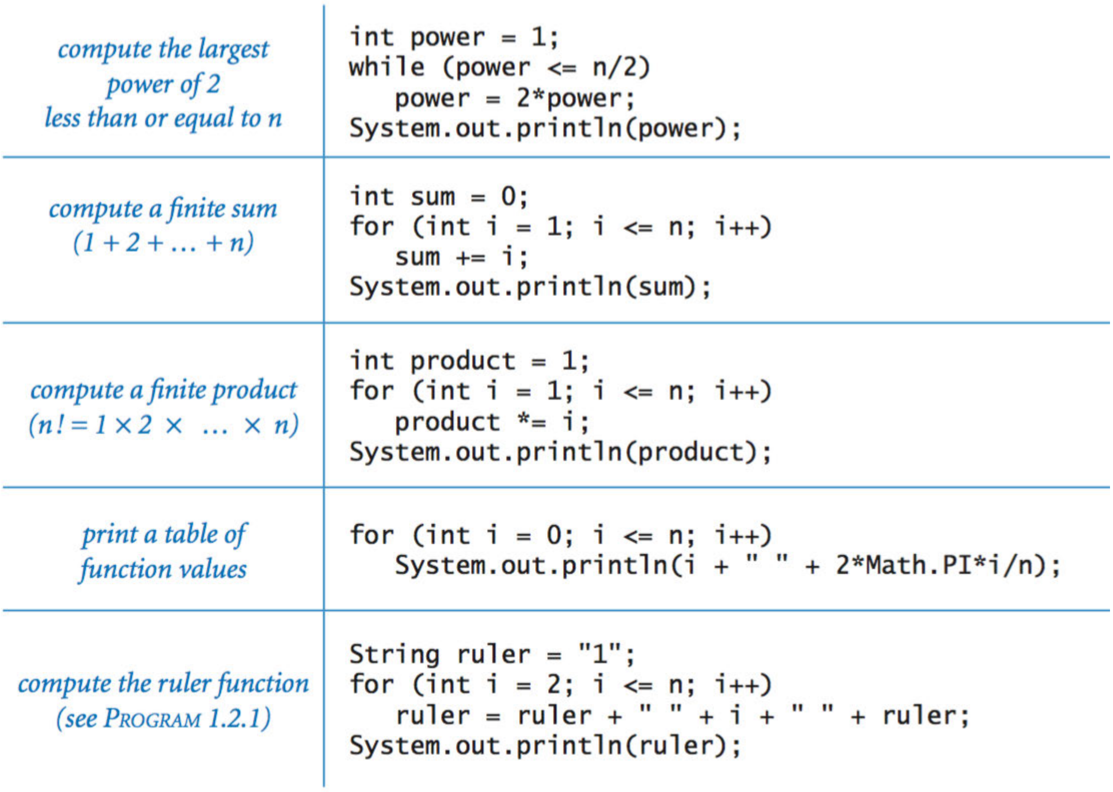
Anatomy of a for loop.



```
int power = 1;
for (int i = 0; i <= n; i++ )
{
   System.out.println(i + " " + power);
   power = 2*power;
}
```

initialize another variable in a separate statement
declare and initialize a loop control variable
loop-continuation condition
increment
body

Loops.



compute the largest power of 2 less than or equal to n	`int power = 1;`
	`while (power <= n/2)`
	`   power = 2*power;`
	`System.out.println(power);`

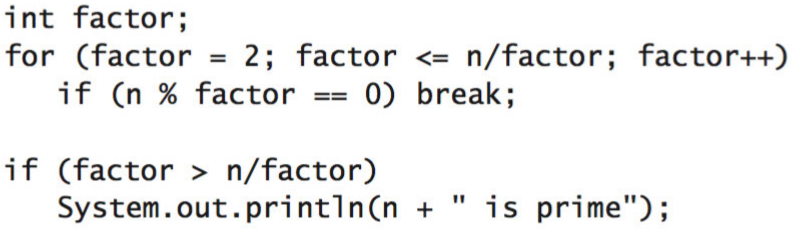
compute a finite sum (1 + 2 + … + n)	`int sum = 0;`
	`for (int i = 1; i <= n; i++)`
	`   sum += i;`
	`System.out.println(sum);`

compute a finite product (n! = 1 × 2 × … × n)	`int product = 1;`
	`for (int i = 1; i <= n; i++)`
	`   product *= i;`
	`System.out.println(product);`

print a table of function values	`for (int i = 0; i <= n; i++)`
	`   System.out.println(i + " " + 2*Math.PI*i/n);`

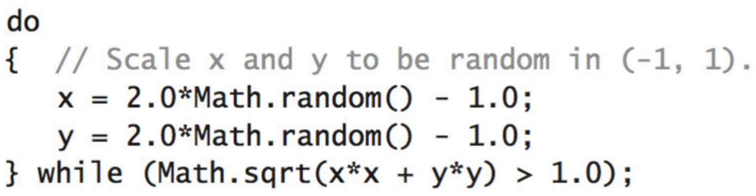
compute the ruler function (see PROGRAM 1.2.1)	`String ruler = "1";`
	`for (int i = 2; i <= n; i++)`
	`   ruler = ruler + " " + i + " " + ruler;`
	`System.out.println(ruler);`

Break statement.

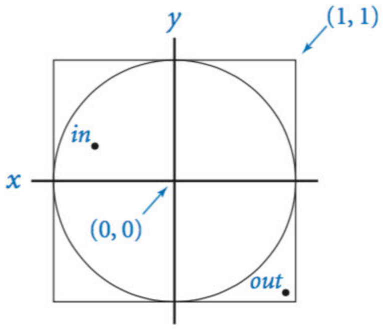


```
int factor;
for (factor = 2; factor <= n/factor; factor++)
   if (n % factor == 0) break;

if (factor > n/factor)
   System.out.println(n + " is prime");
```

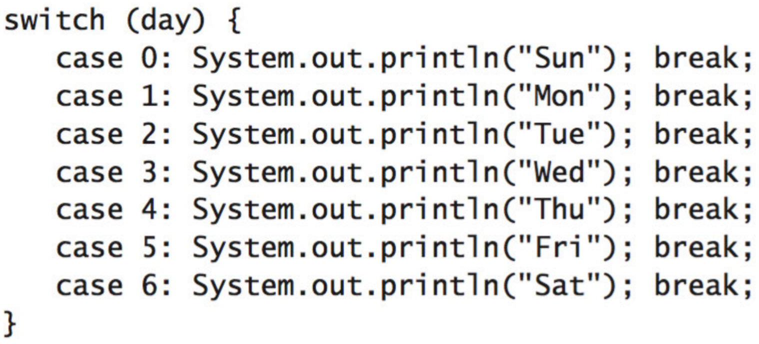
Do-while loop.



```
do
{  // Scale x and y to be random in (-1, 1).
   x = 2.0*Math.random() - 1.0;
   y = 2.0*Math.random() - 1.0;
} while (Math.sqrt(x*x + y*y) > 1.0);
```

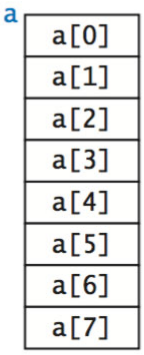


Switch statement.

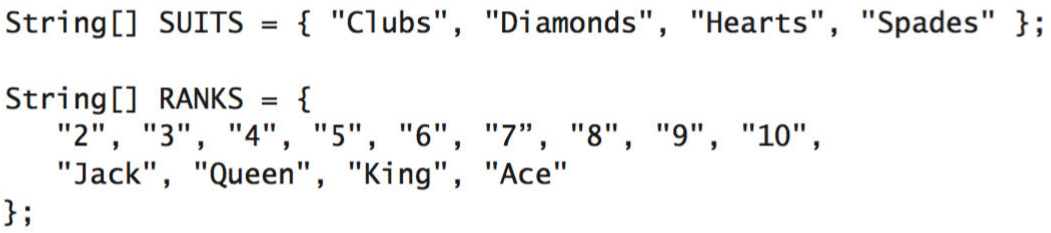


```
switch (day) {
   case 0: System.out.println("Sun"); break;
   case 1: System.out.println("Mon"); break;
   case 2: System.out.println("Tue"); break;
   case 3: System.out.println("Wed"); break;
   case 4: System.out.println("Thu"); break;
   case 5: System.out.println("Fri"); break;
   case 6: System.out.println("Sat"); break;
}
```

Arrays.

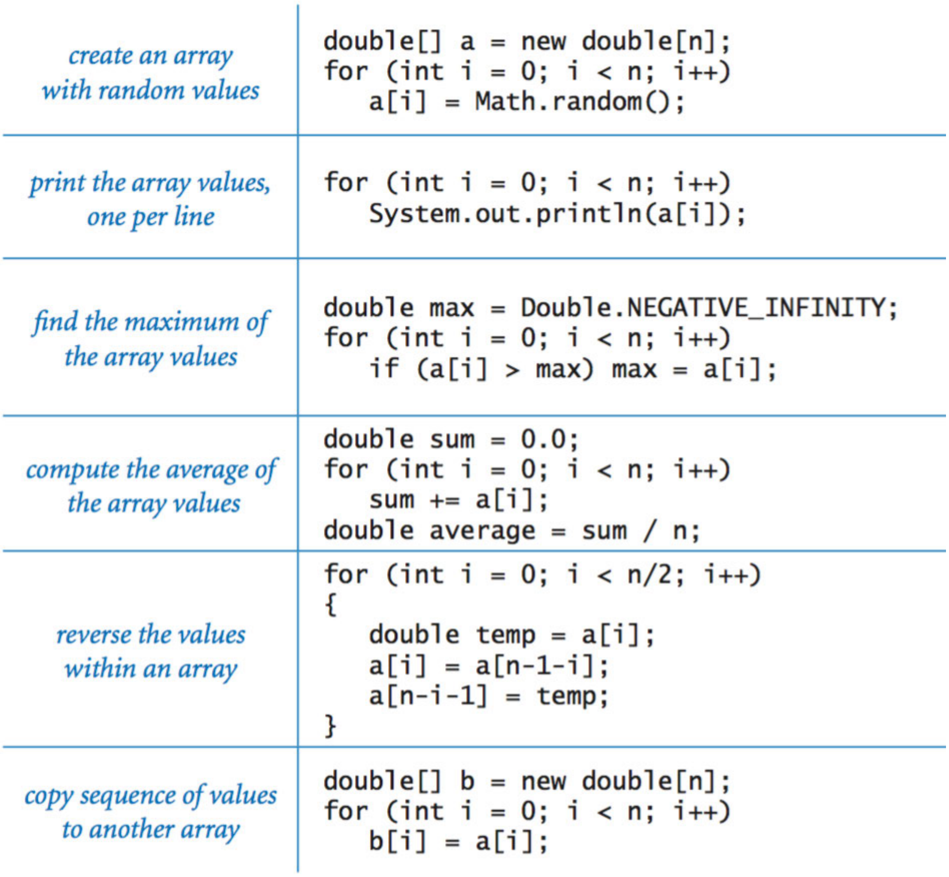


Inline array initialization.



```
String[] SUITS = { "Clubs", "Diamonds", "Hearts", "Spades" };

String[] RANKS = {
   "2", "3", "4", "5", "6", "7", "8", "9", "10",
   "Jack", "Queen", "King", "Ace"
};
```

Typical array-processing code.



create an array with random values	`double[] a = new double[n];`
	`for (int i = 0; i < n; i++)`
	`   a[i] = Math.random();`

print the array values, one per line	`for (int i = 0; i < n; i++)`
	`   System.out.println(a[i]);`

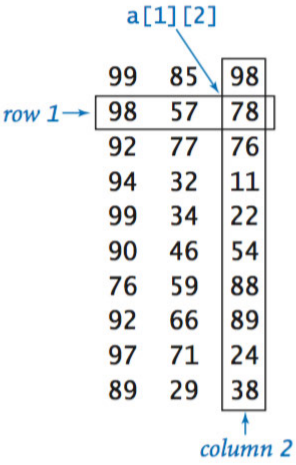
find the maximum of the array values	`double max = Double.NEGATIVE_INFINITY;`
	`for (int i = 0; i < n; i++)`
	`   if (a[i] > max) max = a[i];`

compute the average of the array values	`double sum = 0.0;`
	`for (int i = 0; i < n; i++)`
	`   sum += a[i];`
	`double average = sum / n;`

reverse the values within an array	`for (int i = 0; i < n/2; i++)`
	`{`
	`   double temp = a[i];`
	`   a[i] = a[n-1-i];`
	`   a[n-i-1] = temp;`
	`}`

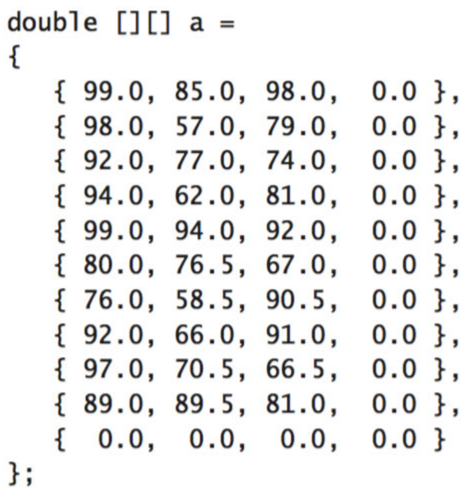
copy sequence of values to another array	`double[] b = new double[n];`
	`for (int i = 0; i < n; i++)`
	`   b[i] = a[i];`

Two-dimensional arrays.



```
99  85  98
98  57  78
92  77  76
94  32  11
99  34  22
90  46  54
76  59  88
92  66  89
97  71  24
89  29  38
```

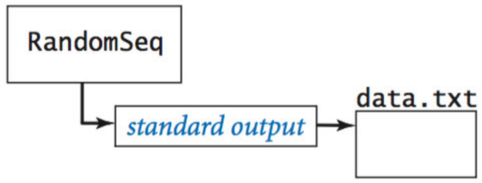
Inline initialization.



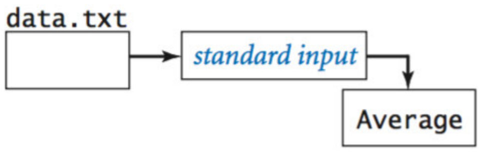
```
double [][] a =
{
   { 99.0, 85.0, 98.0,  0.0 },
   { 98.0, 57.0, 79.0,  0.0 },
   { 92.0, 77.0, 74.0,  0.0 },
   { 94.0, 62.0, 81.0,  0.0 },
   { 99.0, 94.0, 92.0,  0.0 },
   { 80.0, 76.5, 67.0,  0.0 },
   { 76.0, 58.5, 90.5,  0.0 },
   { 92.0, 66.0, 91.0,  0.0 },
   { 97.0, 70.5, 66.5,  0.0 },
   { 89.0, 89.5, 81.0,  0.0 },
   {  0.0,  0.0,  0.0,  0.0 }
};
```

Redirection and piping.

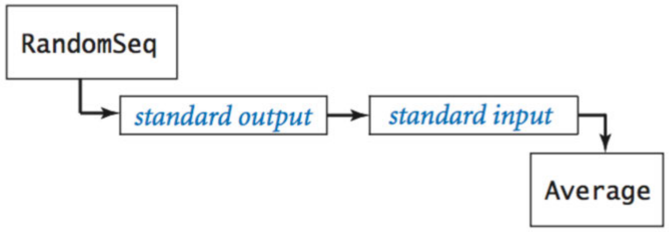
```
% java RandomSeq 1000 > data.txt
```



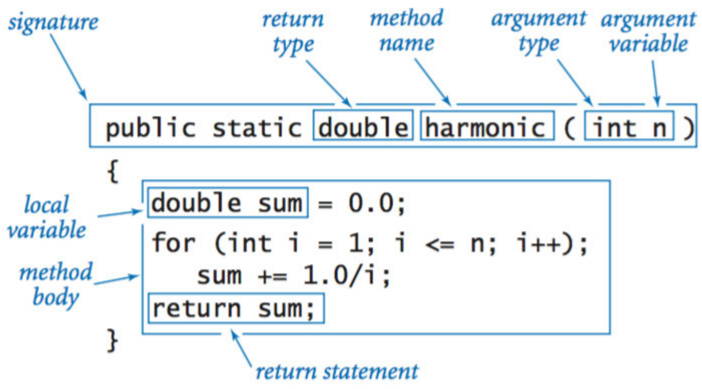
```
% java Average < data.txt
```



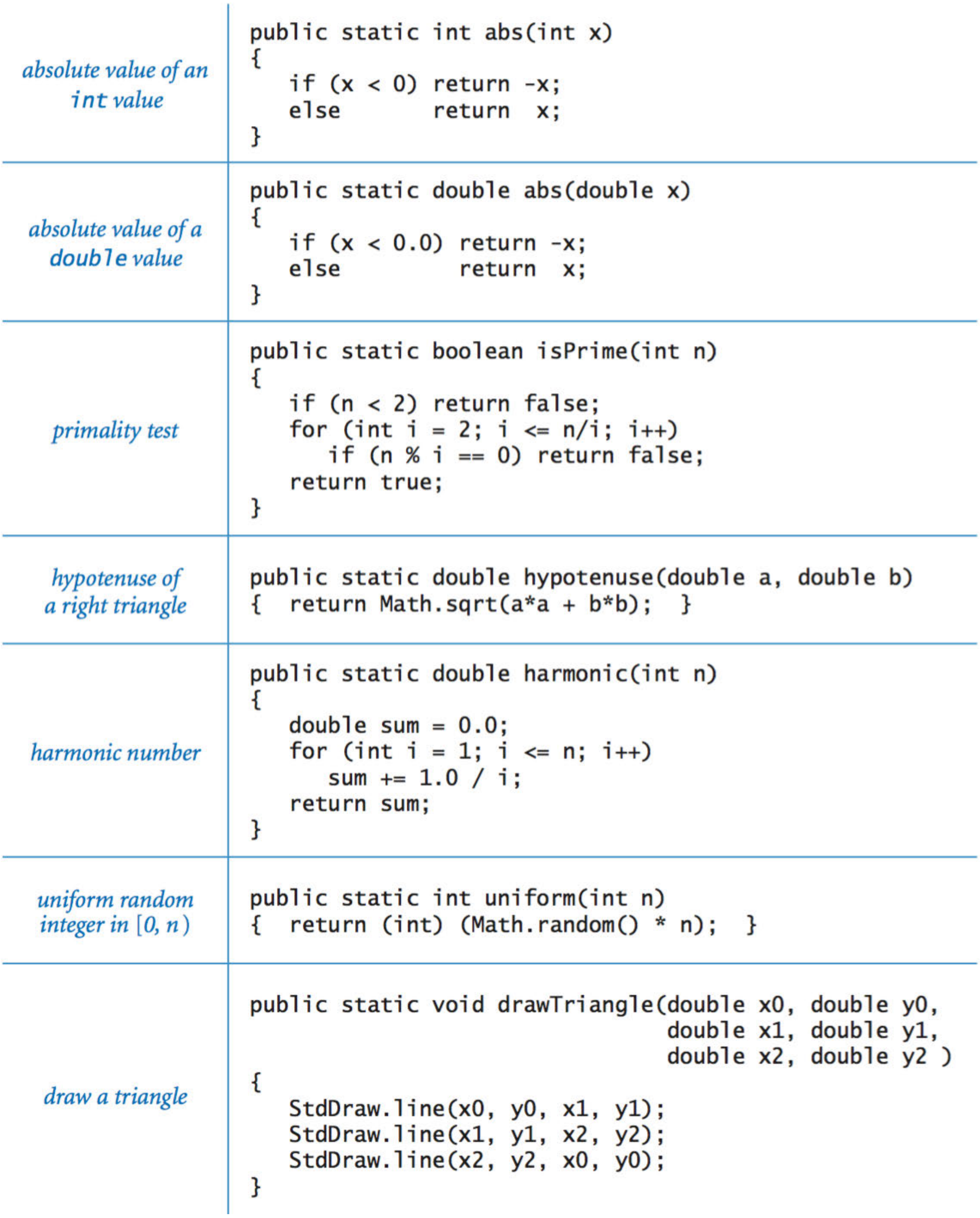
```
% java RandomSeq 1000 | java Average
```



Functions.



```
public static double harmonic ( int n )
{
   double sum = 0.0;
   for (int i = 1; i <= n; i++);
      sum += 1.0/i;
   return sum;
}
```



absolute value of an `int` *value*

```
public static int abs(int x)
{
   if (x < 0) return -x;
   else       return  x;
}
```

absolute value of a `double` *value*

```
public static double abs(double x)
{
   if (x < 0.0) return -x;
   else         return  x;
}
```

primality test

```
public static boolean isPrime(int n)
{
   if (n < 2) return false;
   for (int i = 2; i <= n/i; i++)
      if (n % i == 0) return false;
   return true;
}
```

hypotenuse of a right triangle

```
public static double hypotenuse(double a, double b)
{  return Math.sqrt(a*a + b*b);  }
```

harmonic number

```
public static double harmonic(int n)
{
   double sum = 0.0;
   for (int i = 1; i <= n; i++)
      sum += 1.0 / i;
   return sum;
}
```

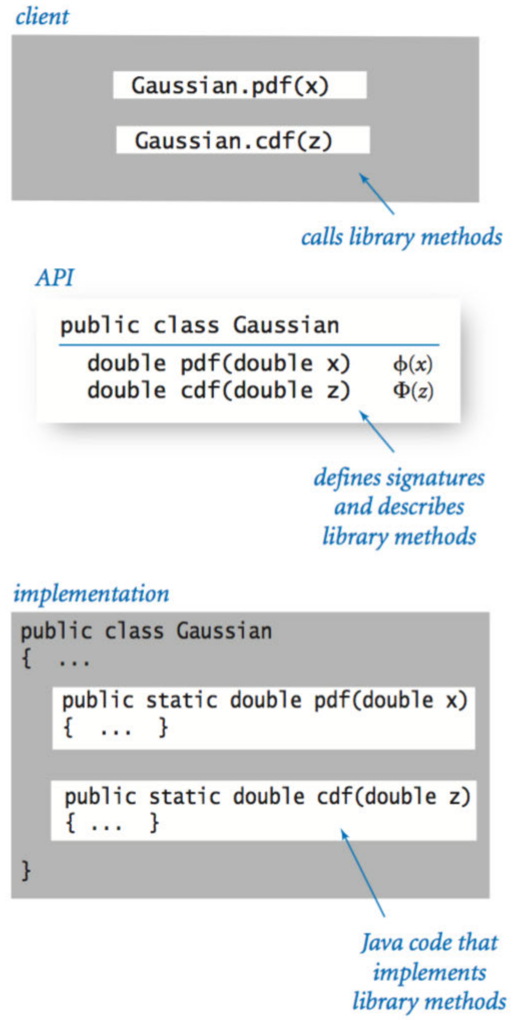
uniform random integer in $[0, n)$

```
public static int uniform(int n)
{  return (int) (Math.random() * n);  }
```

draw a triangle

```
public static void drawTriangle(double x0, double y0,
                                double x1, double y1,
                                double x2, double y2 )
{
   StdDraw.line(x0, y0, x1, y1);
   StdDraw.line(x1, y1, x2, y2);
   StdDraw.line(x2, y2, x0, y0);
}
```

Libraries of functions.



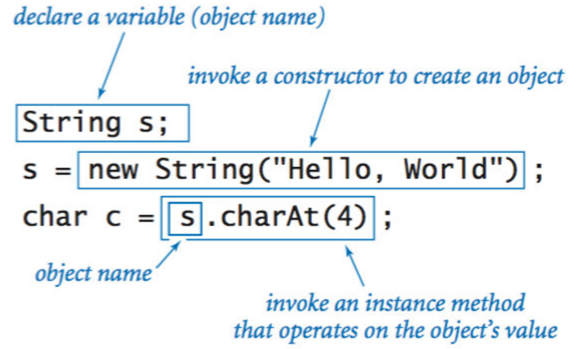
client
calls library methods
API
φ(x)
Φ(z)
defines signatures and describes library methods
implementation
Java code that implements library methods

```
Gaussian.pdf(x)

Gaussian.cdf(z)
```

```
public class Gaussian
   double pdf(double x)
   double cdf(double z)
```

```
public class Gaussian
{  ...
   public static double pdf(double x)
   {  ...  }

   public static double cdf(double z)
   { ...  }
}
```

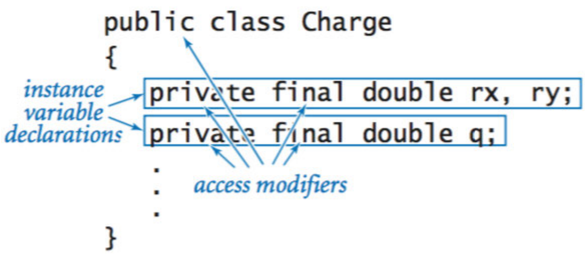
Using an object.



declare a variable (object name)
invoke a constructor to create an object
object name
invoke an instance method that operates on the object's value

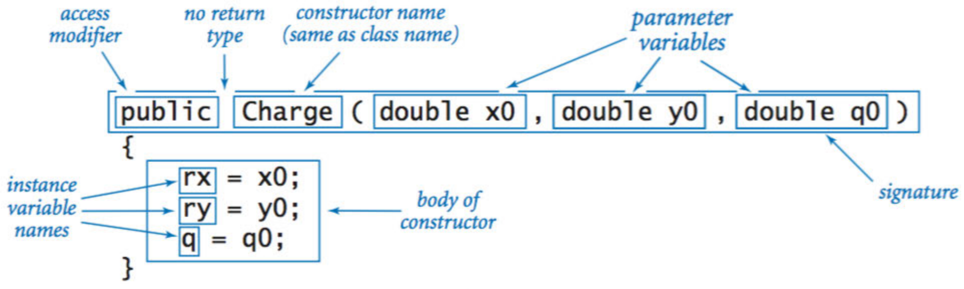
```
String s;
s = new String("Hello, World") ;
char c = s.charAt(4) ;
```

Instance variables.



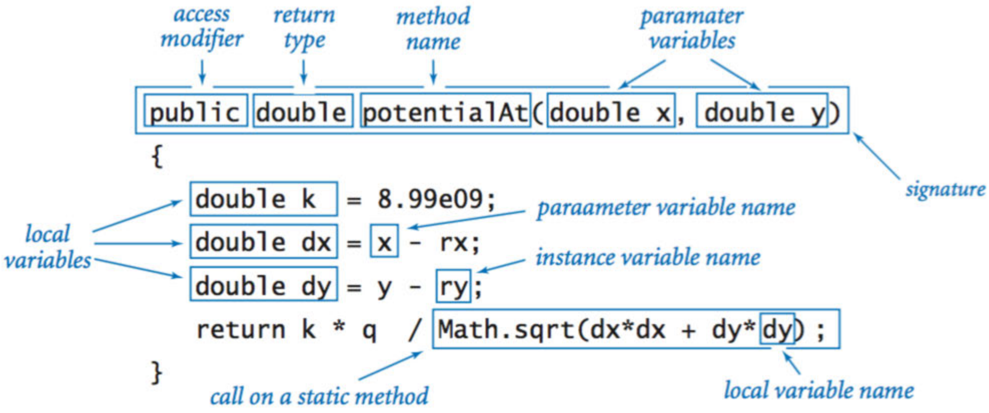
```
public class Charge
{
   private final double rx, ry;
   private final double q;
   .
   .
   .
}
```

Constructors.



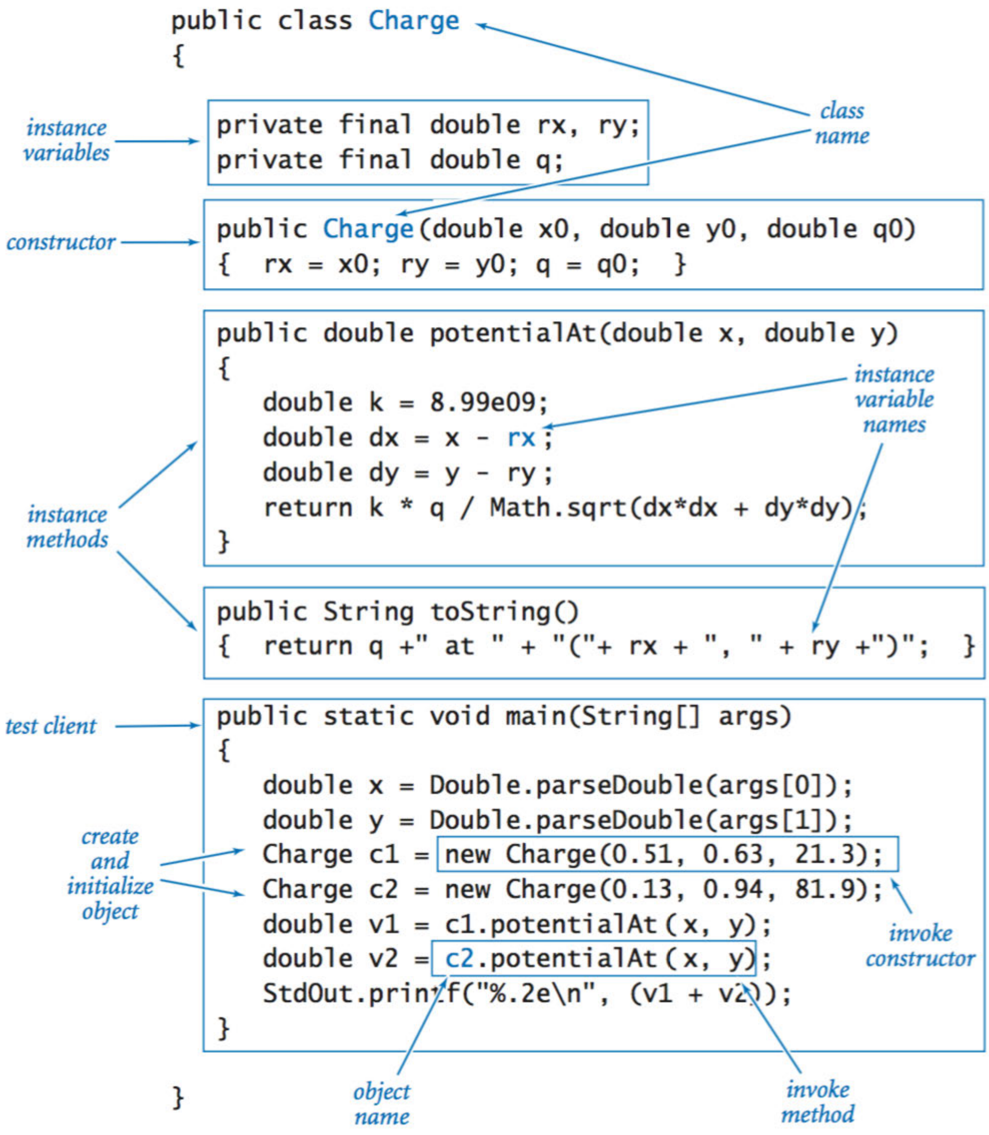
```
public Charge ( double x0 , double y0 , double q0 )
{
   rx = x0;
   ry = y0;
   q = q0;
}
```

Instance methods.



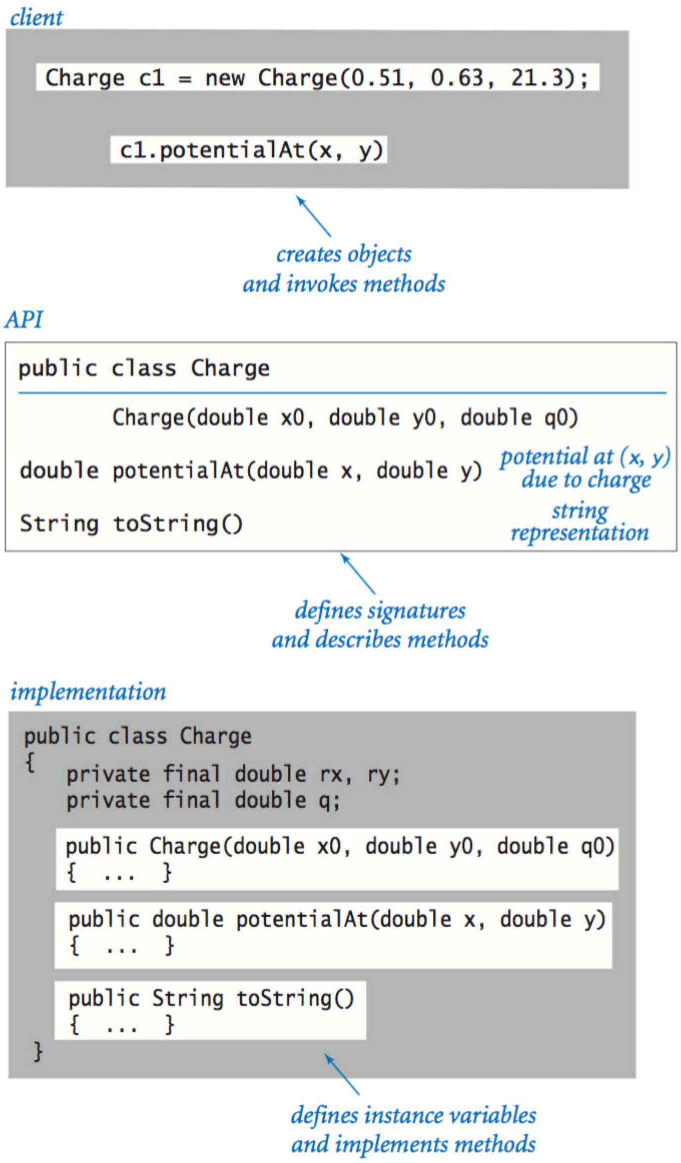
```
public double potentialAt(double x, double y)
{
   double k  = 8.99e09;
   double dx = x - rx;
   double dy = y - ry;
   return k * q  / Math.sqrt(dx*dx + dy*dy) ;
}
```

Classes.



```
public class Charge
{
   private final double rx, ry;
   private final double q;

   public Charge(double x0, double y0, double q0)
   {  rx = x0; ry = y0; q = q0;  }

   public double potentialAt(double x, double y)
   {
      double k = 8.99e09;
      double dx = x - rx;
      double dy = y - ry;
      return k * q / Math.sqrt(dx*dx + dy*dy);
   }

   public String toString()
   {  return q +" at " + "("+ rx + ", " + ry +")";  }

   public static void main(String[] args)
   {
      double x = Double.parseDouble(args[0]);
      double y = Double.parseDouble(args[1]);
      Charge c1 = new Charge(0.51, 0.63, 21.3);
      Charge c2 = new Charge(0.13, 0.94, 81.9);
      double v1 = c1.potentialAt(x, y);
      double v2 = c2.potentialAt(x, y);
      StdOut.printf("%.2e\n", (v1 + v2));
   }

}
```

Object-oriented libraries.



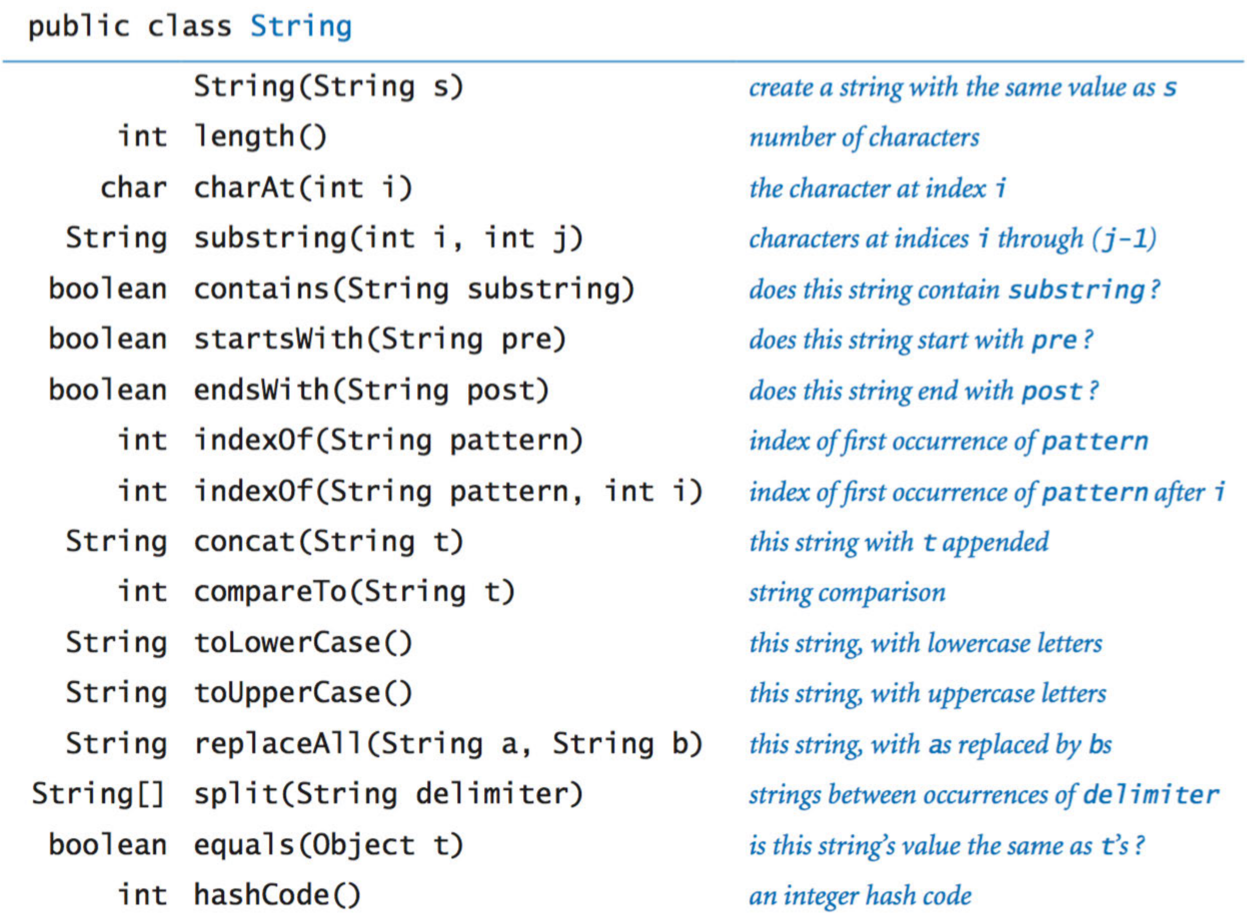
client
creates objects and invokes methods
API
potential at (x, y) due to charge
string representation
defines signatures and describes methods
implementation
defines instance variables and implements methods

```
Charge c1 = new Charge(0.51, 0.63, 21.3);

c1.potentialAt(x, y)
```

```
public class Charge
        Charge(double x0, double y0, double q0)
double  potentialAt(double x, double y)
String  toString()
```

```
public class Charge
{  private final double rx, ry;
   private final double q;

   public Charge(double x0, double y0, double q0)
   {  ...  }

   public double potentialAt(double x, double y)
   {  ...  }

   public String toString()
   {  ...  }
}
```

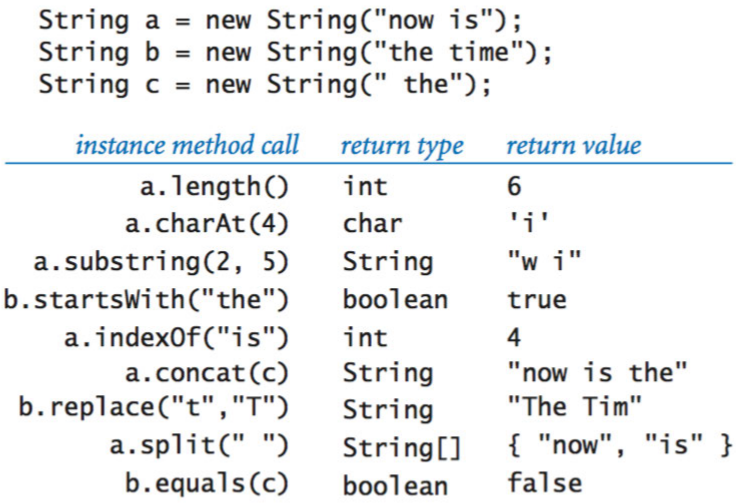
Java's String data type.



public class String

	String(String s)	*create a string with the same value as* s
int	length()	*number of characters*
char	charAt(int i)	*the character at index* i
String	substring(int i, int j)	*characters at indices* i *through* (j-1)
boolean	contains(String substring)	*does this string contain* substring*?*
boolean	startsWith(String pre)	*does this string start with* pre*?*
boolean	endsWith(String post)	*does this string end with* post*?*
int	indexOf(String pattern)	*index of first occurrence of* pattern
int	indexOf(String pattern, int i)	*index of first occurrence of* pattern *after* i
String	concat(String t)	*this string with* t *appended*
int	compareTo(String t)	*string comparison*
String	toLowerCase()	*this string, with lowercase letters*
String	toUpperCase()	*this string, with uppercase letters*
String	replaceAll(String a, String b)	*this string, with* as *replaced by* bs
String[]	split(String delimiter)	*strings between occurrences of* delimiter
boolean	equals(Object t)	*is this string's value the same as* t*'s?*
int	hashCode()	*an integer hash code*

The full java.lang.String API.



```
String a = new String("now is");
String b = new String("the time");
String c = new String(" the");
```

instance method call	*return type*	*return value*
a.length()	int	6
a.charAt(4)	char	'i'
a.substring(2, 5)	String	"w i"
b.startsWith("the")	boolean	true
a.indexOf("is")	int	4
a.concat(c)	String	"now is the"
b.replace("t","T")	String	"The Tim"
a.split(" ")	String[]	{ "now", "is" }
b.equals(c)	boolean	false