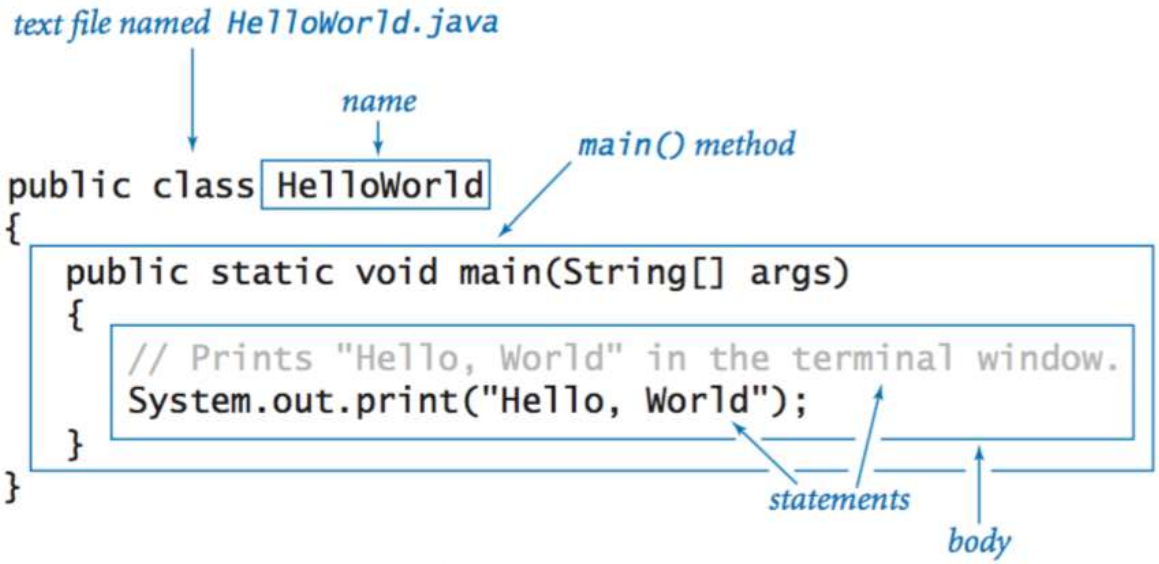
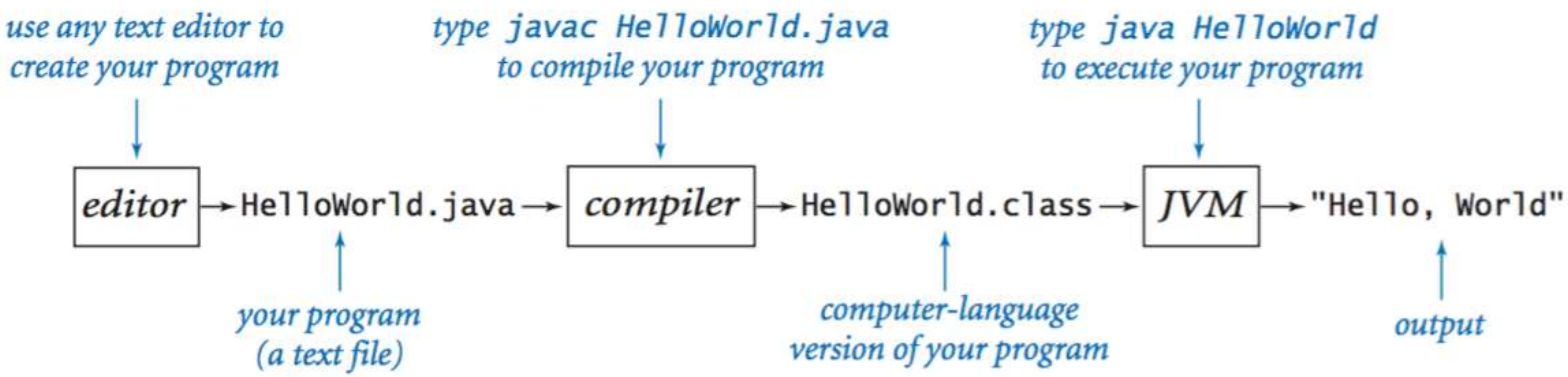


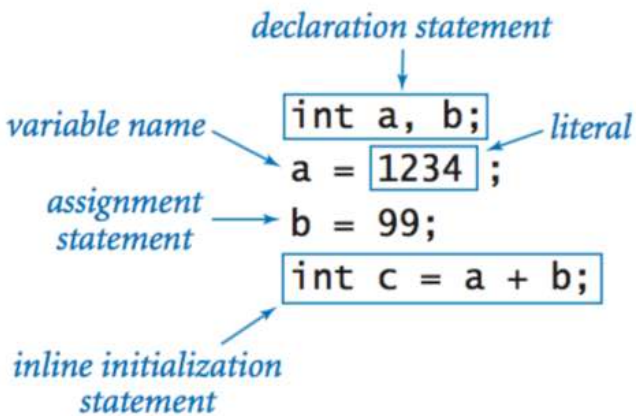
Editing, compiling, and executing.



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"`

Declaration and assignment statements.



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* b*th 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.

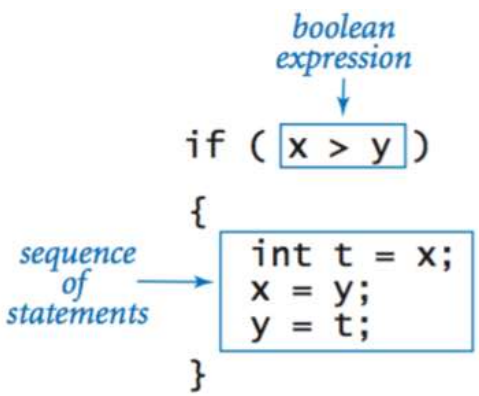
Java library calls.

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 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;`
	`}`

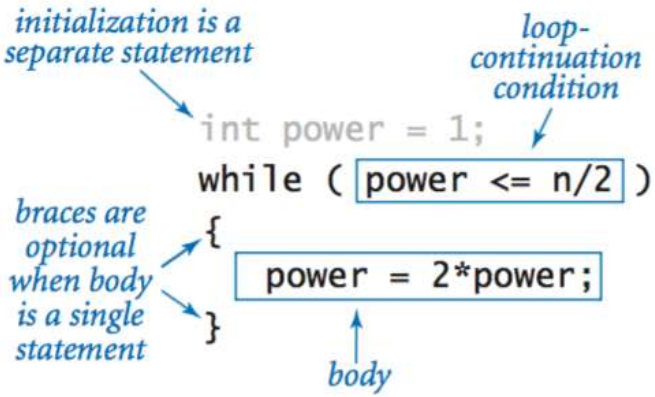
maximum of x and y	`if (x > y) max = x;` `else  max = y;`
error check for division operation	`if (den == 0) System.out.println("Division by zero");` `else  System.out.println("Quotient = " + num/den);`

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);`
	`}`

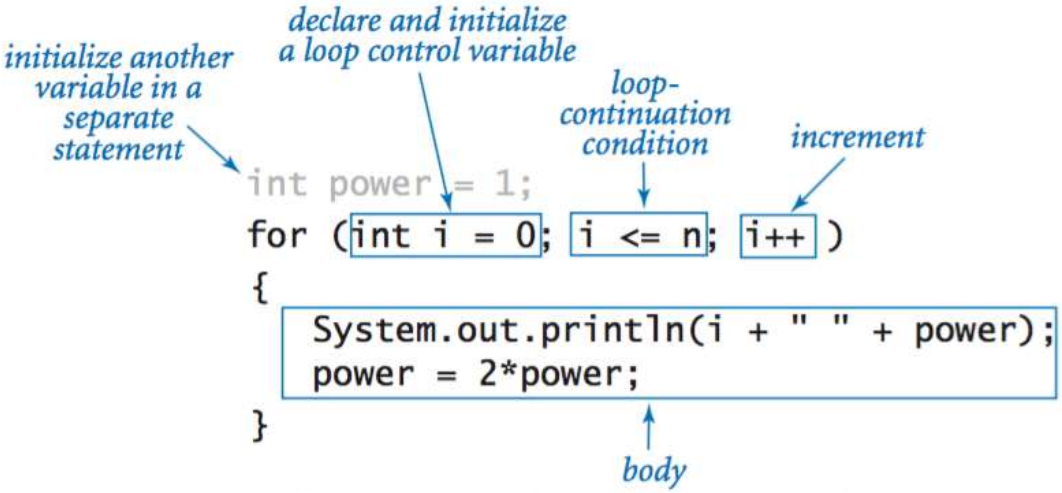
Nested if-else statement.

```
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.



Anatomy of a for loop.


initialize another variable in a separate statement
declare and initialize a loop control variable
loop-continuation condition
increment
int power = 1;
for (int i = 0; i <= n; i++)
{
System.out.println(i + " " + power);
power = 2*power;
}
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 $(1 + 2 + \ldots + n)$	`int sum = 0;`
	`for (int i = 1; i <= n; i++)`
	`  sum += i;`
	`System.out.println(sum);`

compute a finite product $(n! = 1 \times 2 \times \ \ldots \ \times n)$	`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);`

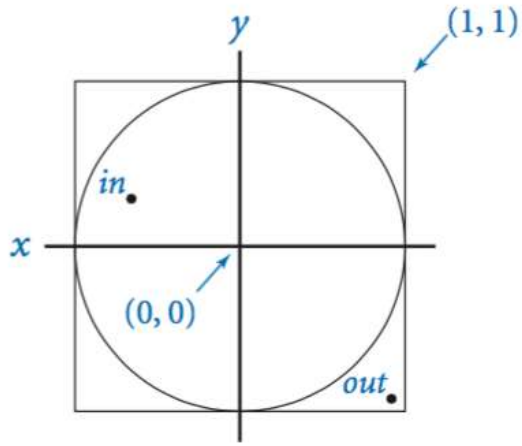
Break statement.

```
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.

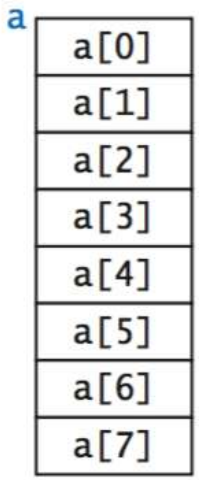
```
do
{  // Scale x and y to be random in (-1, 1).
   x = 2.0*Math.random() - 1.0;
   y = 2.0*Math.random() - 1.0;
} while (Math.sqrt(x*x + y*y) > 1.0);
```



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]);`

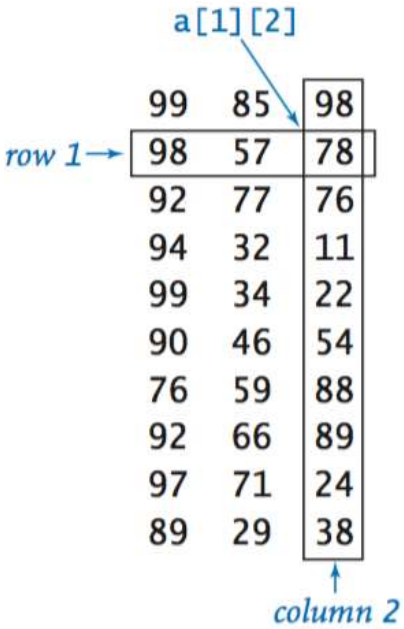
find the maximum of the array values	`double max = Double.NEGATIVE_INFINITY;`
	`for (int i = 0; i < n; i++)`
	`  if (a[i] > max) max = a[i];`

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.



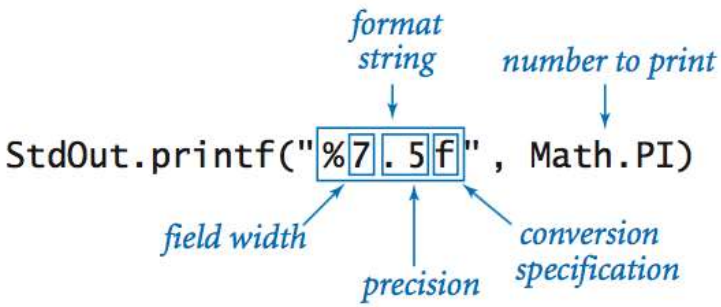
Inline initialization.

```
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 }
};
```

Our standard output library.

public class StdOut	
void print(String s)	*print* `s` *to standard output*
void println(String s)	*print* `s` *and a newline to standard output*
void println()	*print a newline to standard output*
void printf(String format, ...)	*print the arguments to standard output, as specified by the format string* `format`

The full StdOut API.



type	*code*	*typical literal*	*sample format strings*	*converted string values for output*
int	d	512	"%14d"	" 512"
			"%-14d"	"512 "
double	f	1595.1680010754388	"%14.2f"	" 1595.17"
			"%.7f"	"1595.1680011"
	e		"%14.4e"	" 1.5952e+03"
String	s	"Hello, World"	"%14s"	" Hello, World"
			"%-14s"	"Hello, World "
			"%-14.5s"	"Hello "
boolean	b	true	"%b"	"true"

Our standard input library.

public class StdIn

methods for reading individual tokens from standard input		
boolean	isEmpty()	*is standard input empty (or only whitespace)?*
int	readInt()	*read a token, convert it to an int, and return it*
double	readDouble()	*read a token, convert it to a double, and return it*
boolean	readBoolean()	*read a token, convert it to a boolean, and return it*
String	readString()	*read a token and return it as a String*
methods for reading characters from standard input		
boolean	hasNextChar()	*does standard input have any remaining characters?*
char	readChar()	*read a character from standard input and return it*
methods for reading lines from standard input		
boolean	hasNextLine()	*does standard input have a next line?*
String	readLine()	*read the rest of the line and return it as a String*
methods for reading the rest of standard input		
int[]	readAllInts()	*read all remaining tokens and return them as an int array*
double[]	readAllDoubles()	*read all remaining tokens and return them as a double array*
boolean[]	readAllBooleans()	*read all remaining tokens and return them as a boolean array*
String[]	readAllStrings()	*read all remaining tokens and return them as a String array*
String[]	readAllLines()	*read all remaining lines and return them as a String array*
String	readAll()	*read the rest of the input and return it as a String*

The full StdIn API.

Our standard drawing library.

public class StdDraw

drawing commands

```
void line(double x0, double y0, double x1, double y1)
void point(double x, double y)
void circle(double x, double y, double radius)
void filledCircle(double x, double y, double radius)
void square(double x, double y, double radius)
void filledSquare(double x, double y, double radius)
void rectangle(double x, double y, double r1, double r2)
void filledRectangle(double x, double y, double r1, double r2)
void polygon(double[] x, double[] y)
void filledPolygon(double[] x, double[] y)
void text(double x, double y, String s)
```

control commands

`void setXscale(double x0, double x1)`	*reset x-scale to (x0, x1)*
`void setYscale(double y0, double y1)`	*reset y-scale to (y0, y1)*
`void setPenRadius(double radius)`	*set pen radius to* `radius`
`void setPenColor(Color color)`	*set pen color to* `color`
`void setFont(Font font)`	*set text font to* `font`
`void setCanvasSize(int w, int h)`	*set canvas size to w-by-h*

void enableDoubleBuffering()	*enable double buffering*
void disableDoubleBuffering()	*disable double buffering*
void show()	*copy the offscreen canvas to the onscreen canvas*
void clear(Color color)	*clear the canvas to color* `color`
void pause(int dt)	*pause* `dt` *milliseconds*
void save(String filename)	*save to a* `.jpg` *or* `.png` *file*

The full StdDraw API.

Our standard audio library.

public class StdAudio

void play(String filename)	*play the given* `.wav` *file*
void play(double[] a)	*play the given sound wave*
void play(double x)	*play sample for 1/44100 second*
void save(String filename, double[] a)	*save to a* `.wav` *file*
double[] read(String filename)	*read from a* `.wav` *file*

The full StdAudio API.

Command line.

```
public class AddInts
{
   public static void main(String[] args)
   {
      int n = Integer.parseInt(args[0]);
      int sum = 0;
      for (int i = 0; i < n; i++)
      {
         int value = StdIn.readInt();
         sum += value;
      }
      StdOut.println("Sum is " + sum);
   }
}
```

parse command-line argument

read from standard input stream

print to standard output stream

command line

command-line argument

```
% java AddInts 4
```

```
144
233
377
1024
```

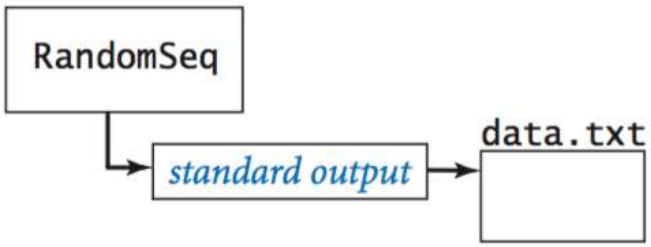
standard input stream

```
Sum is 1778
```

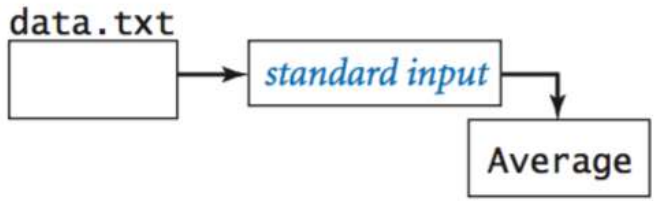
standard output stream

Redirection and piping.

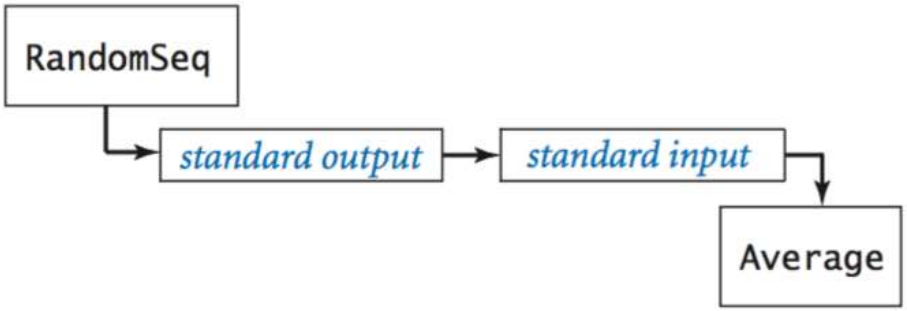
```
% java RandomSeq 1000 > data.txt
```



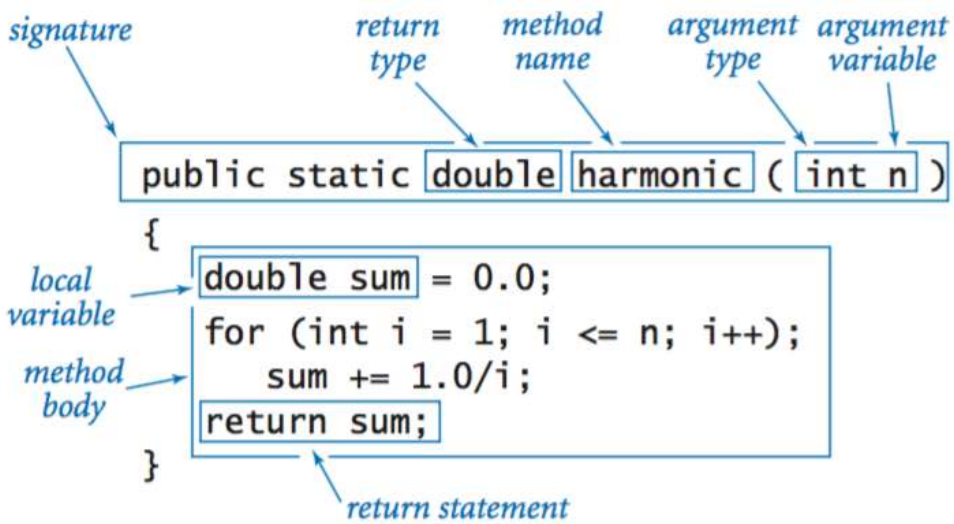

```
% java Average < data.txt
```



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



Functions.



absolute value of an `int` *value*	`public static int abs(int x)`
	`{`
	`  if (x < 0) return -x;`
	`  else  return  x;`
	`}`

absolute value of a	`public static double abs(double x)`
	`{`
	`  if (x < 0.0) return -x;`

double value	`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

```
Gaussian.pdf(x)

Gaussian.cdf(z)
```

calls library methods

API

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

defines signatures and describes library methods

implementation

```
public class Gaussian
{  ...

   public static double pdf(double x)
   {  ...  }

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

}
```

Java code that implements library methods

Our standard random library.

```
public class StdRandom
```

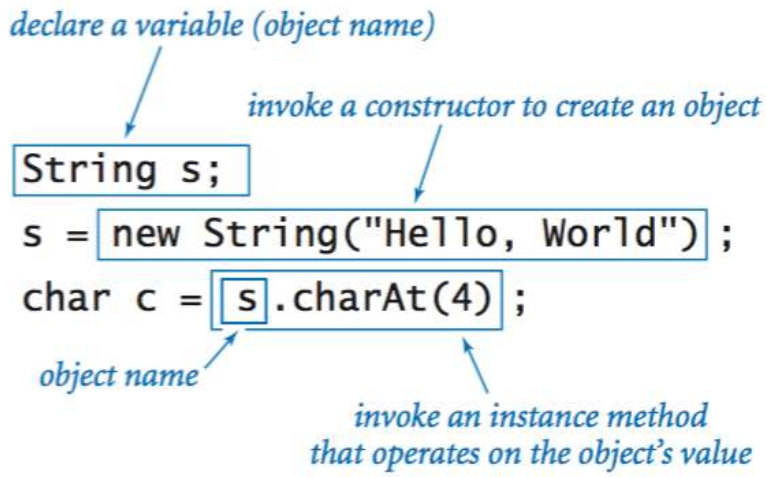
void	setSeed(long seed)	*set the seed for reproducible results*
int	uniform(int n)	*integer between 0 and n-1*
double	uniform(double lo, double hi)	*real between lo and hi*
boolean	bernoulli(double p)	*true with probability p*
double	gaussian()	*normal, mean 0, standard deviation 1*
double	gaussian(double mu, double sigma)	*normal, mean mu, standard deviation sigma*
int	discrete(double[] probabilities)	*i with probability probabilities[i]*
void	shuffle(double[] a)	*randomly shuffle the array a[]*

Our standard statistics library.

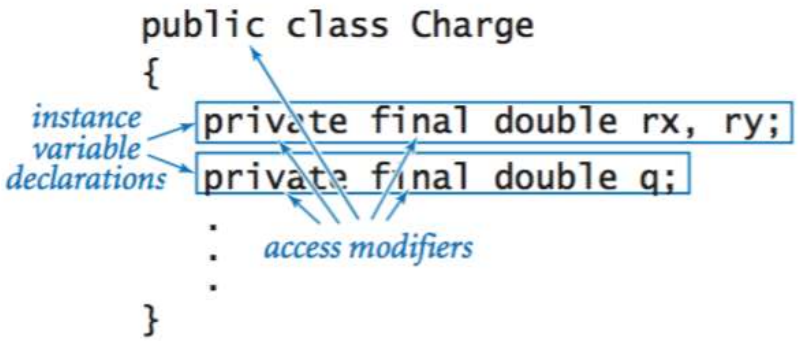
```
public class StdStats
```

double	max(double[] a)	*largest value*
double	min(double[] a)	*smallest value*
double	mean(double[] a)	*average*
double	var(double[] a)	*sample variance*
double	stddev(double[] a)	*sample standard deviation*
double	median(double[] a)	*median*
void	plotPoints(double[] a)	*plot points at (i, a[i])*
void	plotLines(double[] a)	*plot lines connecting (i, a[i])*
void	plotBars(double[] a)	*plot bars to points at (i, a[i])*

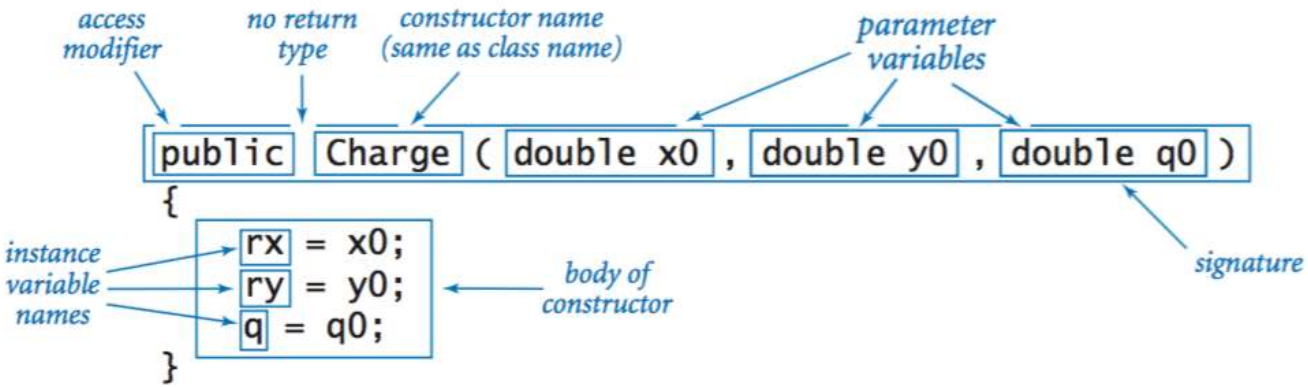
Using an object.



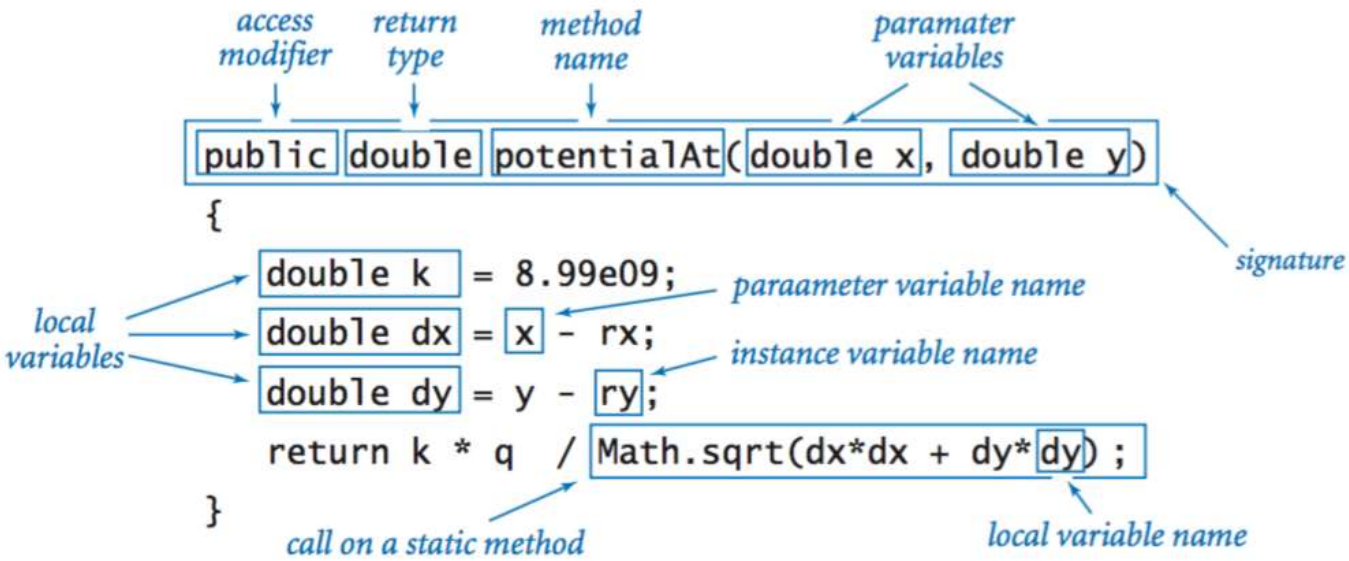
Instance variables.



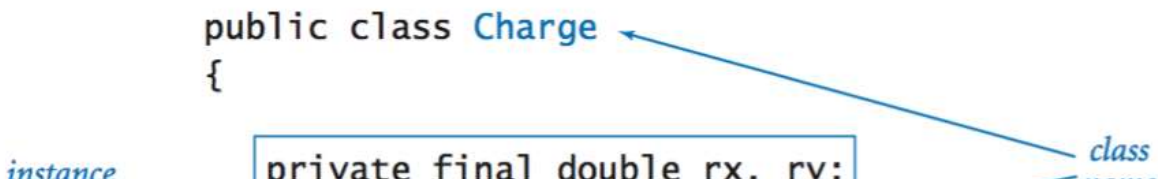
Constructors.



Instance methods.



Classes.



instance variables

```
private final double q;
```

name

constructor

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

instance variable names

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);
}
```

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

test client

create and initialize object

invoke constructor

object name

invoke method

```
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

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

c1.potentialAt(x, y)
```

creates objects and invokes methods

API

```
public class Charge
------------------------------------------------------------
        Charge(double x0, double y0, double q0)
double potentialAt(double x, double y)      potential at (x, y) due to charge
String toString()                           string representation
```

defines signatures and describes methods

implementation

```
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()
   {  ...  }
```

```
}
```

defines instance variables and implements methods

Java's String data type.

```
public class String
```

	`String(String s)`	*create a string with the same value as* `s`
	`String(char[] a)`	*create a string that represents the same sequence of characters as in* `a[]`
`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 prefix)`	*does this string start with* `prefix`*?*
`boolean`	`endsWith(String postfix)`	*does this string end with* `postfix`*?*
`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`	`replace(String a, String b)`	*this string, with* `a`*s replaced by* `b`*s*
		this string, with leading and trailing

String	trim()	*this string, with leading and trailing whitespace removed*
boolean	matches(String regexp)	*is this string matched by the regular expression?*
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 Time"
a.split(" ")	String[]	{ "now", "is" }
b.equals(c)	boolean	false

Java's Color data type.

public class java.awt.Color

	Color(int r, int g, int b)	
int	getRed()	*red intensity*
int	getGreen()	*green intensity*
int	getBlue()	*blue intensity*
Color	brighter()	*brighter version of this color*
Color	darker()	*darker version of this color*
String	toString()	*string representation of this color*
boolean	equals(Object c)	*is this color's value the same as* c *?*

The full java.awt.Color API.

Our input library.

```
public class In
```

	`In()`	*create an input stream from standard input*
	`In(String name)`	*create an input stream from a file or website*
instance methods that read individual tokens from the input stream		
`boolean`	`isEmpty()`	*is standard input empty (or only whitespace)?*
`int`	`readInt()`	*read a token, convert it to an `int`, and return it*
`double`	`readDouble()`	*read a token, convert it to a `double`, and return it*
	`...`	
instance methods that read characters from the input stream		
`boolean`	`hasNextChar()`	*does standard input have any remaining characters?*
`char`	`readChar()`	*read a character from standard input and return it*
instance methods that read lines from the input stream		
`boolean`	`hasNextLine()`	*does standard input have a next line?*
`String`	`readLine()`	*read the rest of the line and return it as a `String`*
instance methods that read the rest of the input stream		
`int[]`	`readAllInts()`	*read all remaining tokens; return as array of integers*
`double[]`	`readAllDoubles()`	*read all remaining tokens; return as array of doubles*
	`...`	

The full In API.

Our output library.

```
public class Out
```

`Out()`	*create an output stream to standard output*
`Out(String name)`	*create an output stream to a file*
`void print(String s)`	*print* `s` *to the output stream*
`void println(String s)`	*print* `s` *and a newline to the output stream*
`void println()`	*print a newline to the output stream*
`void printf(String format, ...)`	*print the arguments to the output stream, as specified by the format string* `format`

The full Out API.

Our picture library.

```
public class Picture
```

`Picture(String filename)`	*create a picture from a file*
`Picture(int w, int h)`	*create a blank* `w`*-by-*`h` *picture*
`int width()`	*return the width of the picture*
`int height()`	*return the height of the picture*
`Color get(int col, int row)`	*return the color of pixel (*`col`*,* `row`*)*
`void set(int col, int row, Color color)`	*set the color of pixel (*`col`*,* `row`*) to* `color`
`void show()`	*display the picture in a window*
`void save(String filename)`	*save the picture to a file*

The full Picture API.

Our stack data type.

```
public class Stack<Item> implements Iterable<Item>
```

	`Stack()`	*create an empty stack*
`boolean`	`isEmpty()`	*is the stack empty?*
`void`	`push(Item item)`	*push an item onto the stack*
`Item`	`pop()`	*return and remove the item that was inserted most recently*
`int`	`size()`	*number of items on stack*

The full Stack API.

Our queue data type.

```
public class Queue<Item> implements Iterable<Item>
```

	`Queue()`	*create an empty queue*
`boolean`	`isEmpty()`	*is the queue empty?*
`void`	`enqueue(Item item)`	*insert an item onto queue*
`Item`	`dequeue()`	*return and remove the item that was inserted least recently*
`int`	`size()`	*number of items on queue*

The full Queue API.

Iterable.

```
import java.util.Iterator;

public class Queue<Item>
      implements Iterable<Item>
{
   private Node first;
   private Node last;
   private class Node
   {
      Item item;
      Node next;
   }
   public void enqueue(Item item)
   ...
   public Item dequeue()
   ...

   public Iterator<Item> iterator()
   {  return new ListIterator();  }

   private class ListIterator
         implements Iterator<Item>
   {
      Node current = first;

      public boolean hasNext()
      {  return current != null;  }

      public Item next()
      {
         Item item = current.item;
         current = current.next;
         return item;
      }

      public void remove()
      { }
   }

   public static void main(String[] args)
   {
      Queue<Integer> queue = new Queue<Integer>();
      while (!StdIn.isEmpty())
```

not in language

promise to implement iterator()

FIFO queue code

implementation for Iterable interface

additional code to make the class iterable

promise to implement hasNext(), next(), and remove()

implementations for Iterator interface

```
        queue.enqueue(StdIn.readInt());
        for (int s : queue)
            StdOut.println(s);
    }
}
```

foreach statement

Our symbol table data type.

```
public class ST<Key extends Comparable<Key>, Value>
```

	`ST()`	*create an empty symbol table*
`void`	`put(Key key, Value val)`	*associate* `val` *with* `key`
`Value`	`get(Key key)`	*value associated with* `key`
`void`	`remove(Key key)`	*remove* `key` *(and its associated value)*
`boolean`	`contains(Key key)`	*is there a value associated with* `key`*?*
`int`	`size()`	*number of key–value pairs*
`Iterable<Key>`	`keys()`	*all keys in the symbol table*

The full ST API.

Our set data type.

```
public class SET<Key extends Comparable<Key>> implements Iterable<Key>
```

	SET()	*create an empty set*
boolean	isEmpty()	*is the set empty?*
void	add(Key key)	*add* key *to the set*
void	remove(Key key)	*remove* key *from set*
boolean	contains(Key key)	*is* key *in the set?*
int	size()	*number of elements in set*

The full SET API.

Our graph data type.

```
public class Graph
```

	Graph()	*create an empty graph*
	Graph(String filename, String delimiter)	*create graph from a file*
void	addEdge(String v, String w)	*add edge* v-w
int	V()	*number of vertices*
int	E()	*number of edges*
Iterable<String>	vertices()	*vertices in the graph*
Iterable<String>	adjacentTo(String v)	*neighbors of* v
int	degree(String v)	*number of neighbors of* v
boolean	hasVertex(String v)	*is* v *a vertex in the graph?*
boolean	hasEdge(String v, String w)	*is* v-w *an edge in the graph?*