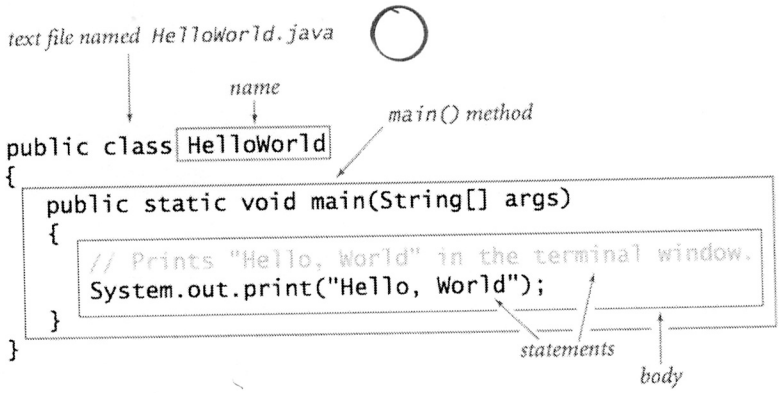
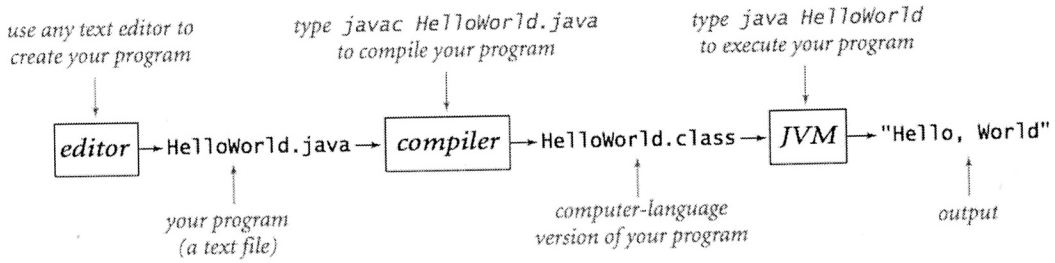
Java Cheat Sheet
Version 2

Source: Princeton University
https://introcs.cs.princeton.edu/java/11cheatsheet



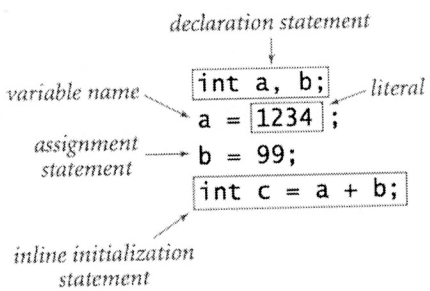
Editing, compiling, and executing.



Built-in data types.

type	*set of values*	*common operators*	*sample literal values*	
`int`	integers	`+ - * / %`	`99 12 2147483647`	primitive data types
`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"`	Built in class

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	`+ -`	`+`	`-`	`*`	`/`	`%`

Integers.

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.

Boolean

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* ~B
`double max(double a, double b)`	*maximum of a and b* ~B
`double min(double a, double b)`	*minimum of a and b* ~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) ~B
`long round(double a)`	*round a to the nearest integer* ~B
`double random()`	*random number in* [0, 1) Does not include one. ~B
`double sqrt(double a)`	*square root of a* ~B
`double E`	*value of e (constant)*
`double PI`	*value of π (constant)*

The full java.lang.Math API.

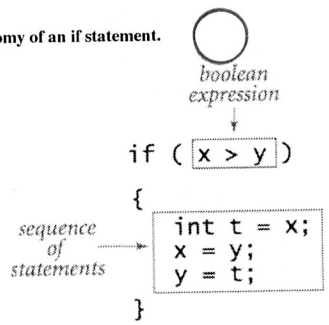
Java library calls.

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)	~B
`Math.round(3.14159)`	`Math`	`long`	`3`	~B
`Math.max(1.0, 9.0)`	`Math`	`double`	`9.0`	~B

Type conversion. – ~B

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

Anatomy of an if statement.



If and if-else statements.

If then else

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);`
	`}`

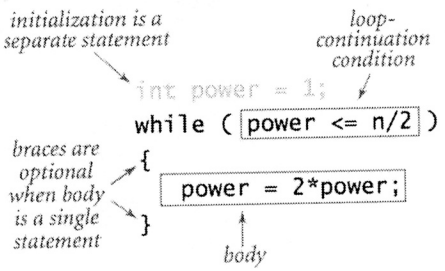
NB
Swops two values around

NB

Nested if-else statement.

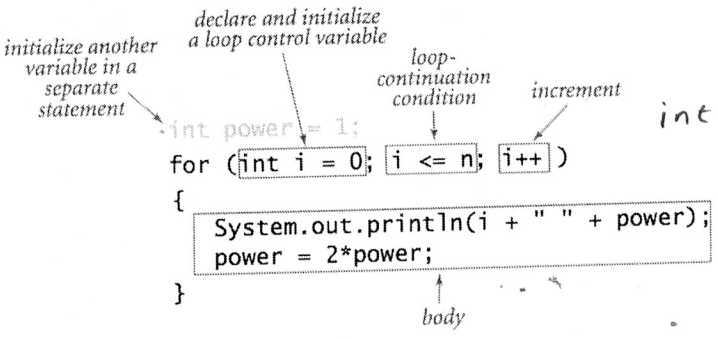
```
if      (income <      0) rate = 0.00;
else if (income <   8925) rate = 0.10;
else if (income <  36250) rate = 0.15;
else if (income <  87850) rate = 0.23;
else if (income < 183250) rate = 0.28;
else if (income < 398350) rate = 0.33;
else if (income < 400000) rate = 0.35;
else                   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

Anatomy of a while loop.



int power = 1;

Anatomy of a for loop.



int power = 1;

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);
```

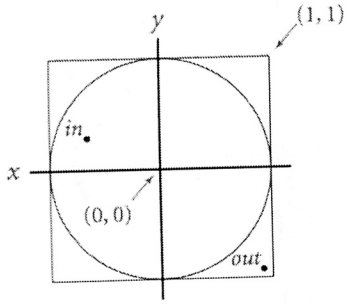
Break statement.

```
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.

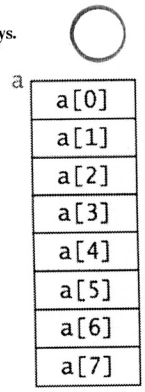
```
do
{  // Scale x and y to be random in (-1, 1).
   x = 2.0*Math.random() - 1.0;
   y = 2.0*Math.random() - 1.0;
} while (Math.sqrt(x*x + y*y) > 1.0);
```



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]);`

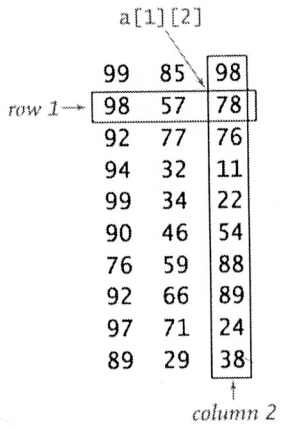
find the maximum of the array values	`double max = Double.NEGATIVE_INFINITY;`
	`for (int i = 0; i < n; i++)`
	`  if (a[i] > max) max = a[i];`

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.



Inline initialization.

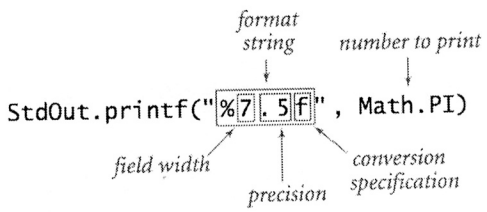
```
double [][] a =
{
   { 99.0, 85.0, 98.0,  0.0 },
   { 98.0, 57.0, 79.0,  0.0 },
   { 92.0, 77.0, 74.0,  0.0 },
   { 94.0, 62.0, 81.0,  0.0 },
   { 99.0, 94.0, 92.0,  0.0 },
   { 80.0, 76.5, 67.0,  0.0 },
   { 76.0, 58.5, 90.5,  0.0 },
   { 92.0, 66.0, 91.0,  0.0 },
   { 97.0, 70.5, 66.5,  0.0 },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



Input - Scanner Class – Methods

String next( ), Boolean hasNext( ), String nextLine( ), void close( ),
String useDelimiter( )
int nextInt( ), long nextLong( ), float nextFloat( ), double nextDouble( )

Input – Scanner Class – Default delimiter

```
import java.util. * ;

String line = "Hello 45 65.9 true";
Scanner scLine = new Scanner(line);

String greet = scLine.next( );
int num = scLine.nextInt( );
double real = scLine.nextDouble( );
boolean ans = scLine.nextBoolean( );
scLine.close( );
```

Input – Scanner Class – Reading Data from a text file

NB: Know your character handling methods !!

Multiple fields and two different delimiters . . . default (the space) and #

```
John Smith#23
Jabulani Nlhapo#45

import java.io. * ;              // To read text files
import java.util. * ;            // For the scanner class

public static void main (String[ ] args) throws IOException
{
        Scanner scFile = new Scanner (new File ("NamesAge.txt") );
        String line = " " , name = " " ;
        int num, sumOfAges = 0 ;

        while (scFile.hasNext( ) )
        {
                line = scFile.nextLine( ) ;
                Scanner scLine = new Scanner (line).useDelimiter("#");
                name = scLine.next( );
                num = scLine.nextInt( );
                scLine.close( );        // close the line for re-use next iteration
                System.out.println(name + "\t" + num);
                sumOfAges = sumOfAges + num;

        }       // end while

        scFile.close( );        // close the file outside the loop
        System.out.println("The sum of their ages is " + sumOfAges);
}       // end main
```

public class StdDraw

Maybe for PAT

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. Maybe for PAT

public class StdAudio

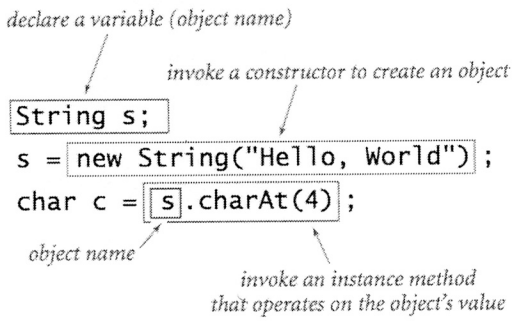
`void play(String filename)`	*play the given .wav file*
`void play(double[] a)`	*play the given sound wave*
`void play(double x)`	*play sample for 1/44100 second*
`void save(String filename, double[] a)`	*save to a .wav file*
`double[] read(String filename)`	*read from a .wav file*

The full StdAudio API.

Command line.

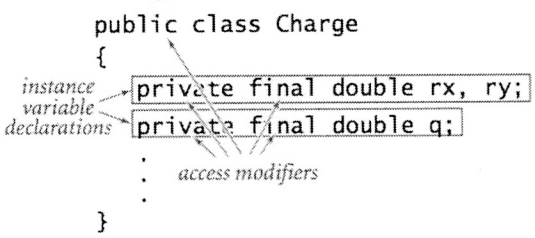
See additional notes pg 2-5

Using an object.

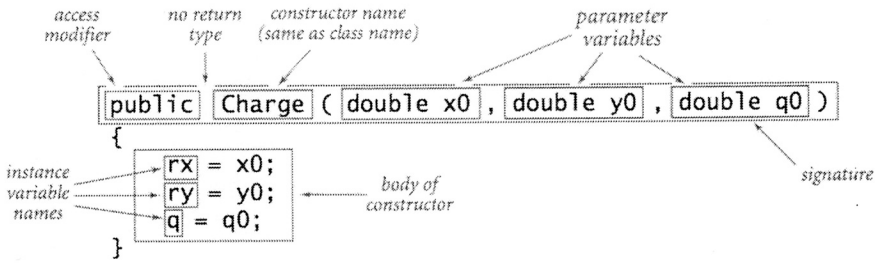


NB

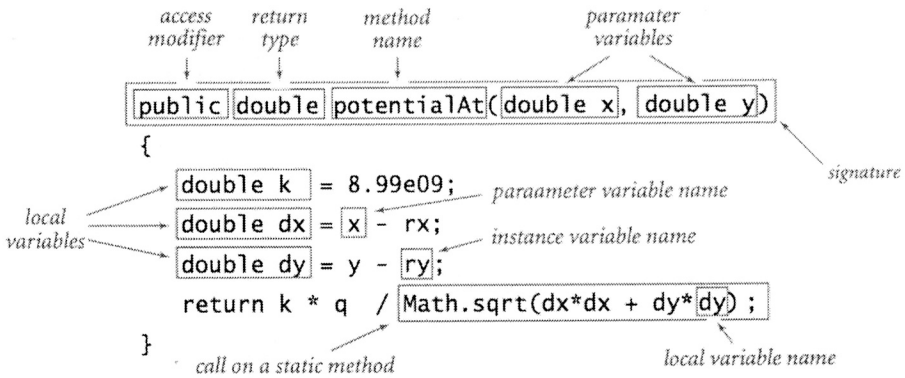
Instance variables.



Constructors.



Instance methods.



Classes.

Declaring variables

- int number;
- String lastName;
- double interestRate;
- int raceOne, raceTwo, raceThree – allowed by not recommended

Variable naming convention

- Must start with a letter
- Combination of letters, numerals, underscores and dollar sign.

```
public class HelloApp
{
```

Area for instance variables, which should be private - (unless you use the word "static" making them available to the whole class.)

```
    public static void main(String[ ] args)
    {
```

Area for local variables - only available to the local method – in this case "main".

```
    }
}
```


15A

1) Declaring variables that are going to be used by the whole class

Class variables – static variables

A variable that any method in a class can access including static methods such as main (this is about avoiding the error message non static variable cannot be referenced from a static context)

Where to place a class variable

- Within the body of the class but not within any of the class methods.
- Not within the main method.
- Must include the reserved word "static" before the variable type

```
public class HelloApp
{
      static String helloMessage;    // static variable
      available to whole class

      public static void main(String[ ] args)
      {
            helloMessage = "Hello World");
            System.out.println(helloMessage);
      }
}
```

2) Variables that are going to be used by a created object for its <u>own</u> use

Instance variables (non static variables)

Variables that are part of an instance of a class (part of a created object) (this is also about avoiding the error message "non static variable cannot be referenced from a static context").

Where to place an instance variable

- Within the body of the class but not within any of the class methods.
- Not within the main method. (Classes used to create further objects seldom have a main method.)
- Does **not** use the reserved word "static" before the variable type.

You cannot use an instance variable in a static method – and this includes the main method. (Static methods are not associated with an instance of a class.)

15B

Run time error below . . . "non static variable cannot be referenced from a static context".

```
public class HelloApp
{
     String helloMessage;      //non static, instance variable.

     public static void main(String[ ] args)
     {
          helloMessage = "Hello World");
          System.out.println(helloMessage);
     }
}
```

The main method is static and therefore cannot access an instance variable.

Example: Instance variables

A class called student. From this class multiple new student objects are created. Each object has the same instance variables to store student details.

```
public class Student {
     // instance variables private to this class.
     private String firstName;
     private String lastName;
     private int daysAbsent;
     private int grade;
     private char class;

     // Constructor
     public Student (String fn, String ln, int da, int gr,
                     char cl) {
          firstName = fn;
          lastName = ln;
          daysAbsent = da;
          grade = gr;
          class = cl;
     } // end constructor

     // no main method. Student class is called by the
      application program where the main method can be found.

     // Getter methods to make the internal data of this
     class available to the rest of the application.

     public String getFirstName( ) {
          return firstName;
     }

     public String getLastName( ) {
          return lastName;
     }
} // end class
```


15c

3) Declaring a local variable

A variable that is declared within the body of a method (including the main method). These variables are only available within that method and cannot be used by other methods or classes.

Where to place a local variable

- Within the body of the class and . . .
- Within the relevant method
    - Within the main method – if relevant

```
public class HelloApp
{
      public static void main(String[ ] args)
      {
            String helloMessage;
            // available only to this method

            helloMessage = "Hello World");
            System.out.println(helloMessage);
      }
}
```

4) Declaring a final variable

A final variable, also called a constant, is a variable whose value you cannot change after it's been initialized. Useful in a program that makes use of the same variable throughout the program i.e. a variable that must not change.

e.g. final int WEEKDAYS = 5; // By convention final variables are all caps

Generally you would like your constant to be available throughout your application therefore final variables are usually *class variables* so we add the static modifier.

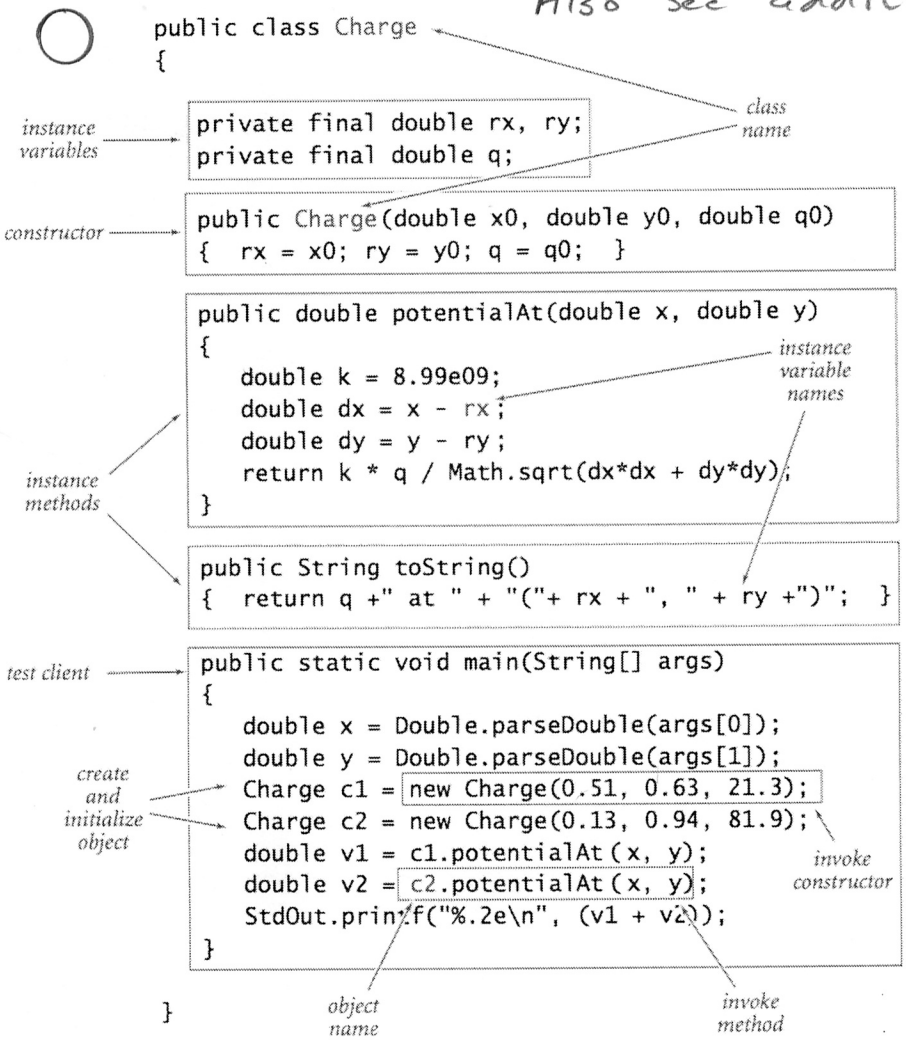
e.g. static final WEEKDAYS = 5;

Where to place a final variable

- A final variable is usually a class or instance variable (but it can be a local variable.
- See notes on class and instance variables.

15 D

Also see additional notes pg 2-5



Object-oriented libraries.

public class String — NB

	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* NB
char	charAt(int i)	*the character at index* i NB
String	substring(int i, int j)	*characters at indices* i *through* (j-1) NB
boolean	contains(String substring)	*does this string contain* substring? NB
boolean	startsWith(String prefix)	*does this string start with* prefix? NB
boolean	endsWith(String postfix)	*does this string end with* postfix? NB
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* NB
String	toLowerCase()	*this string, with lowercase letters* NB
String	toUpperCase()	*this string, with uppercase letters* NB
String	replace(String a, String b)	*this string, with* a*s replaced by* b*s*
String	trim()	*this string, with leading and trailing whitespace removed* NB
boolean	matches(String regexp)	*is this string matched by the regular expression?*
String[]	split(String delimiter)	*strings between occurrences of* delimiter NB
boolean	equals(Object t)	*is this string's value the same as* t*'s?* NB
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

NB

NB

Java's Color data type.