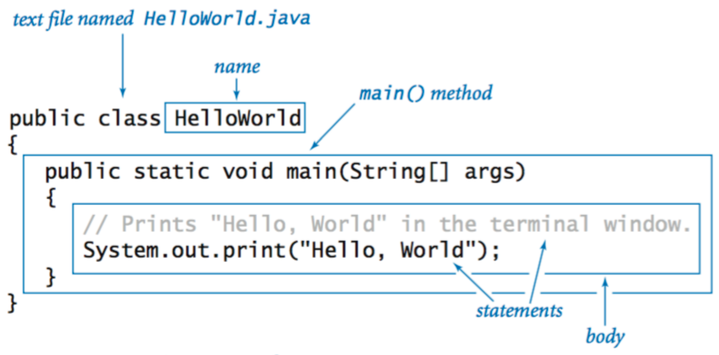
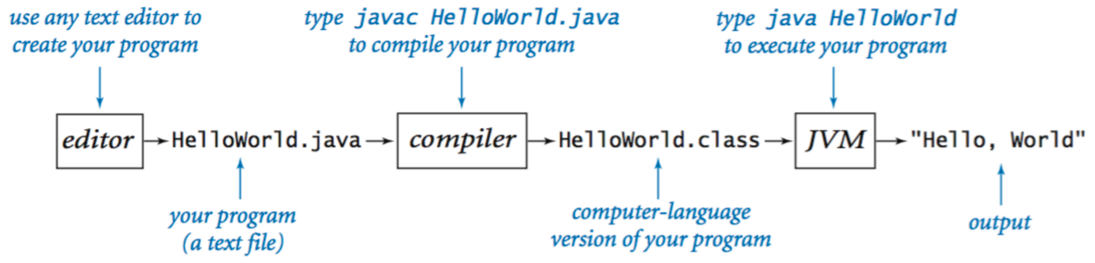
## Hello, World




## Editing, compiling, and executing




| 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 |

| values | true or false |
|---|---|
| literals | true false |
| operations | and or not |
| operators | && \|\| ! |

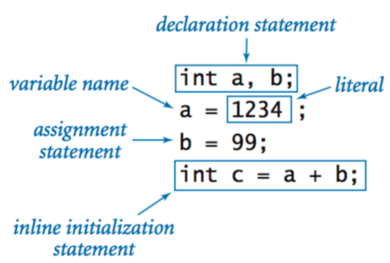
| a | !a |
|---|---|
| true | false |
| false | true |

| a | b | a && b | a \|\| b |
|---|---|---|---|
| false | false | false | false |
| false | true | false | true |
| true | false | false | true |
| true | true | true | true |

## 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




## Naming variables
- Can't start with a digit
- Contains only letters, digits, and underscore
- No keywords

## Real vs integer division
- Can use type conversion to avoid

```
double a = 7, b = 2;
int x = 5, y = 2;

System.out.println ( a / b ) ;     // prints: 3.5
System.out.println ( a / y ) ;     // prints: 3.5
System.out.println ( x / b ) ;     // prints: 2.5
System.out.println ( x / y) ;      // prints: 2
```

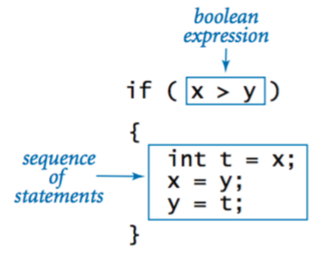
## Prefix and Postfix increments

```
int x = 5, y = 2;
int z;
z = ++y;   // first increments y then uses y
System.out.println(z);  // prints 3
System.out.println(y);  // prints 3
z = x++;   // first uses x then increments x
System.out.println(z);  // prints 5
System.out.println(x);  // prints 6
```

## Comparison Operators

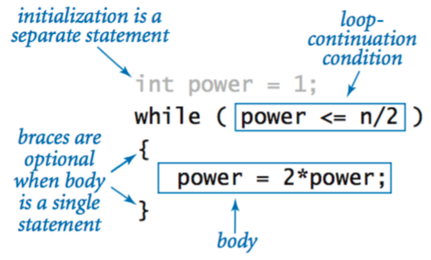
| op | meaning | true | false |
|---|---|---|---|
| == | equal | 2 == 2 | 2 == 3 |
| != | not equal | 3 != 2 | 2 != 2 |
| < | less than | 2 < 13 | 2 < 2 |
| <= | less than or equal | 2 <= 2 | 3 <= 2 |
| > | greater than | 13 > 2 | 2 > 13 |
| >= | greater than or equal | 3 >= 2 | 2 >= 3 |

## If 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;
```

## While loop

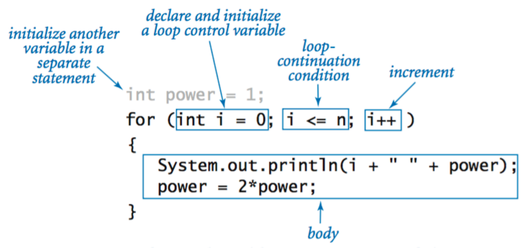



## Do-while loop

```
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);
```

## Loop statements
- break (breaks out of loop)
- continue (skips to end of loop)

## For loop




```
for (int num : numberArray)
{
   num += 5;
}
```

```
// for each loop
for (int i: someArray) {}
```

## Error types
- Compilation/syntax errors (picked up by the computer, code can't run)
- Logical errors (bugs, syntax correct but program doesn't do what you want)
  - e.g. using + instead of –
  - runtime errors (out of bound, null pointer exception)
    - cause crash at run time

## 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;
}
```

default: System.out.println("Try input again");

## Array (fixed length)
- 1D array, list of elements
- 2D array, table of elements

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

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 }
};
```

```
// Initializing
type[] varName= new type[size];
// Declaring
type[] varName= new type[]{values1, value2,...};
// Initializing
datatype[][] varName  =  new dataType[row][col];
// Declaring
datatype[][] varName  =  {{value1, value2....},{value1,
value2....}..};
```

int[] myArray = new int[5];
int[] myArray = {'h', 'e', 'l', 'l', 'o'};
myArray[0] contains 'h'
myArray.length
creates an array of object references

java.util.Arrays
- Arrays.binarySearch(arr, 23)
- Arrays.equals(arr1, arr2)
- Arrays.fill(arr, 23)
- Arrays.sort(arr)

## Parsing string for digit or digit to string
- String s = String.valueOf(value)

| | | |
|---|---|---|
| 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) |

## Type Conversion (int) (double) (type)
- int y = (int) x;
- Can also just initialize variable of desired type with other variable

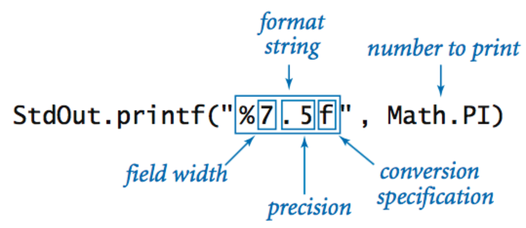
| expression | expression type | expression value |
|---|---|---|
| (1 + 2 + 3 + 4) / 4.0 | double | 2.5 |
| Math.sqrt(4) | double | 2.0 |
| "1234" + 99 | String | "123499" |
| 11 * 0.25 | double | 2.75 |
| (int) 11 * 0.25 | double | 2.75 |
| 11 * (int) 0.25 | int | 0 |
| (int) (11 * 0.25) | int | 2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

## 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 |

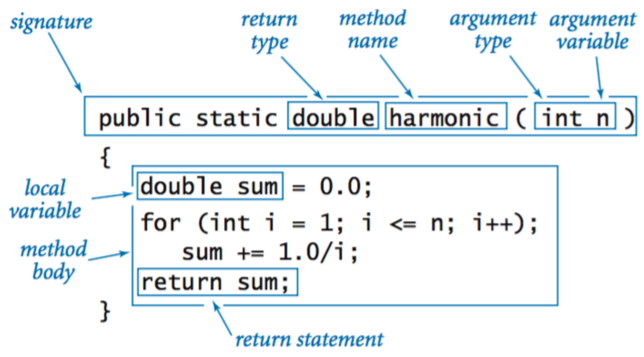
void System.out.printf(String s) print formatted string

## StdOut API and formatted strings




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

## Functions




## 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 |

## 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]) |

## ASCII Table

| Dec | Hx | Oct | Char | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 000 | NUL (null) | 32 | 20 | 040 |   | Space | 64 | 40 | 100 | @ | @ | 96 | 60 | 140 | ` | ` |
| 1 | 1 | 001 | SOH (start of heading) | 33 | 21 | 041 | ! | ! | 65 | 41 | 101 | A | A | 97 | 61 | 141 | a | a |
| 2 | 2 | 002 | STX (start of text) | 34 | 22 | 042 | " | " | 66 | 42 | 102 | B | B | 98 | 62 | 142 | b | b |
| 3 | 3 | 003 | ETX (end of text) | 35 | 23 | 043 | # | # | 67 | 43 | 103 | C | C | 99 | 63 | 143 | c | c |
| 4 | 4 | 004 | EOT (end of transmission) | 36 | 24 | 044 | $ | $ | 68 | 44 | 104 | D | D | 100 | 64 | 144 | d | d |
| 5 | 5 | 005 | ENQ (enquiry) | 37 | 25 | 045 | % | % | 69 | 45 | 105 | E | E | 101 | 65 | 145 | e | e |
| 6 | 6 | 006 | ACK (acknowledge) | 38 | 26 | 046 | & | & | 70 | 46 | 106 | F | F | 102 | 66 | 146 | f | f |
| 7 | 7 | 007 | BEL (bell) | 39 | 27 | 047 | ' | ' | 71 | 47 | 107 | G | G | 103 | 67 | 147 | g | g |
| 8 | 8 | 010 | BS (backspace) | 40 | 28 | 050 | ( | ( | 72 | 48 | 110 | H | H | 104 | 68 | 150 | h | h |
| 9 | 9 | 011 | TAB (horizontal tab) | 41 | 29 | 051 | ) | ) | 73 | 49 | 111 | I | I | 105 | 69 | 151 | i | i |
| 10 | A | 012 | LF (NL line feed, new line) | 42 | 2A | 052 | * | * | 74 | 4A | 112 | J | J | 106 | 6A | 152 | j | j |
| 11 | B | 013 | VT (vertical tab) | 43 | 2B | 053 | + | + | 75 | 4B | 113 | K | K | 107 | 6B | 153 | k | k |
| 12 | C | 014 | FF (NP form feed, new page) | 44 | 2C | 054 | , | , | 76 | 4C | 114 | L | L | 108 | 6C | 154 | l | l |
| 13 | D | 015 | CR (carriage return) | 45 | 2D | 055 | - | - | 77 | 4D | 115 | M | M | 109 | 6D | 155 | m | m |
| 14 | E | 016 | SO (shift out) | 46 | 2E | 056 | . | . | 78 | 4E | 116 | N | N | 110 | 6E | 156 | n | n |
| 15 | F | 017 | SI (shift in) | 47 | 2F | 057 | / | / | 79 | 4F | 117 | O | O | 111 | 6F | 157 | o | o |
| 16 | 10 | 020 | DLE (data link escape) | 48 | 30 | 060 | 0 | 0 | 80 | 50 | 120 | P | P | 112 | 70 | 160 | p | p |
| 17 | 11 | 021 | DC1 (device control 1) | 49 | 31 | 061 | 1 | 1 | 81 | 51 | 121 | Q | Q | 113 | 71 | 161 | q | q |
| 18 | 12 | 022 | DC2 (device control 2) | 50 | 32 | 062 | 2 | 2 | 82 | 52 | 122 | R | R | 114 | 72 | 162 | r | r |
| 19 | 13 | 023 | DC3 (device control 3) | 51 | 33 | 063 | 3 | 3 | 83 | 53 | 123 | S | S | 115 | 73 | 163 | s | s |
| 20 | 14 | 024 | DC4 (device control 4) | 52 | 34 | 064 | 4 | 4 | 84 | 54 | 124 | T | T | 116 | 74 | 164 | t | t |
| 21 | 15 | 025 | NAK (negative acknowledge) | 53 | 35 | 065 | 5 | 5 | 85 | 55 | 125 | U | U | 117 | 75 | 165 | u | u |
| 22 | 16 | 026 | SYN (synchronous idle) | 54 | 36 | 066 | 6 | 6 | 86 | 56 | 126 | V | V | 118 | 76 | 166 | v | v |
| 23 | 17 | 027 | ETB (end of trans. block) | 55 | 37 | 067 | 7 | 7 | 87 | 57 | 127 | W | W | 119 | 77 | 167 | w | w |
| 24 | 18 | 030 | CAN (cancel) | 56 | 38 | 070 | 8 | 8 | 88 | 58 | 130 | X | X | 120 | 78 | 170 | x | x |
| 25 | 19 | 031 | EM (end of medium) | 57 | 39 | 071 | 9 | 9 | 89 | 59 | 131 | Y | Y | 121 | 79 | 171 | y | y |
| 26 | 1A | 032 | SUB (substitute) | 58 | 3A | 072 | : | : | 90 | 5A | 132 | Z | Z | 122 | 7A | 172 | z | z |
| 27 | 1B | 033 | ESC (escape) | 59 | 3B | 073 | ; | ; | 91 | 5B | 133 | [ | [ | 123 | 7B | 173 | { | { |
| 28 | 1C | 034 | FS (file separator) | 60 | 3C | 074 | < | < | 92 | 5C | 134 | \ | \ | 124 | 7C | 174 | | | \| |
| 29 | 1D | 035 | GS (group separator) | 61 | 3D | 075 | = | = | 93 | 5D | 135 | ] | ] | 125 | 7D | 175 | } | } |
| 30 | 1E | 036 | RS (record separator) | 62 | 3E | 076 | > | > | 94 | 5E | 136 | ^ | ^ | 126 | 7E | 176 | ~ | ~ |
| 31 | 1F | 037 | US (unit separator) | 63 | 3F | 077 | ? | ? | 95 | 5F | 137 | _ | _ | 127 | 7F | 177 | &#127; | DEL |

## Character addition etc.

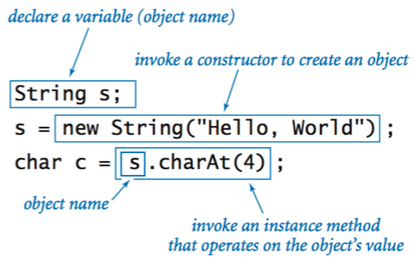
```
char ltr;
ltr = 'A' + 1;
System.out.println(ltr);
```

B

## Operator Precedence

| Level | Operator | Description | Associativity |
|---|---|---|---|
| 16 | []<br>.<br>() | access array element<br>access object member<br>parentheses | left to right |
| 15 | ++<br>-- | unary post-increment<br>unary post-decrement | not associative |
| 14 | ++<br>--<br>+<br>-<br>!<br>~ | unary pre-increment<br>unary pre-decrement<br>unary plus<br>unary minus<br>unary logical NOT<br>unary bitwise NOT | right to left |
| 13 | ()<br>new | cast<br>object creation | right to left |
| 12 | * / % | multiplicative | left to right |
| 11 | + -<br>+ | additive<br>string concatenation | left to right |
| 10 | << >><br>>>> | shift | left to right |
| 9 | < <=<br>> >=<br>instanceof | relational | not associative |
| 8 | ==<br>!= | equality | left to right |
| 7 | & | bitwise AND | left to right |
| 6 | ^ | bitwise XOR | left to right |
| 5 | \| | bitwise OR | left to right |
| 4 | && | logical AND | left to right |
| 3 | \|\| | logical OR | left to right |
| 2 | ?: | ternary | right to left |
| 1 | = += -=<br>*= /= %=<br>&= ^= \|=<br><<= >>= >>>= | assignment | right to left |

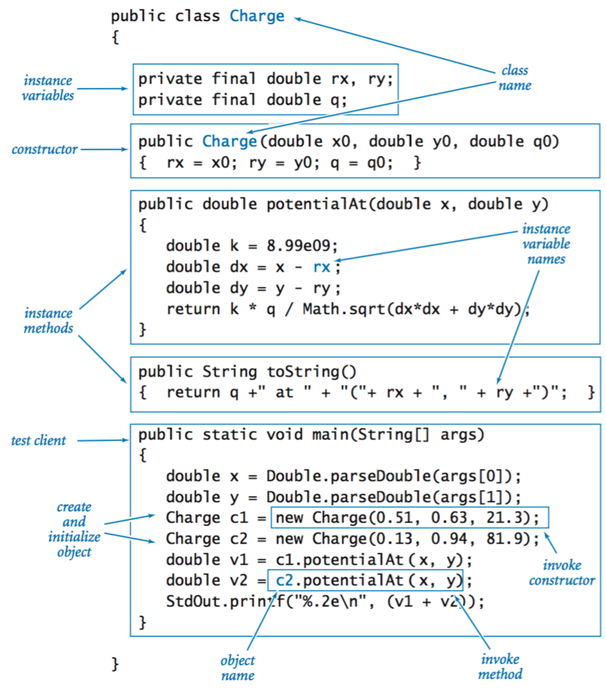
## Using an object (instance of a class)




## Getters, setters, and toString

- Getters return info from object
- Setters receive info to set variables
- toString() allows instances to be printed

## Classes




## User input

```
// Using BufferReader
BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in));
String name = reader.readLine();
// Using Scanner
Scanner in = new Scanner(System.in);
String s = in.nextLine();
int a = in.nextInt();
// Using Console
String name = System.console().readLine();
```

## Scanner

```
Scanner scan = new Scanner(System.in);
System.out.println("Please enter your name: ");
String name = scan.nextLine();
```

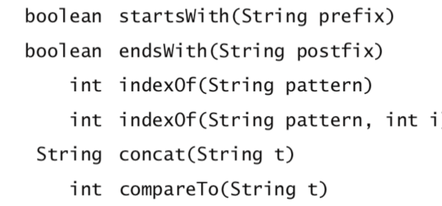
The scanner can also reads from a string:

```
String input = "1 2 orange apple ";
Scanner s = new Scanner(input);
System.out.println(s.nextInt());
System.out.println(s.nextInt());
System.out.println(s.next());
System.out.println(s.next());
s.close();
```

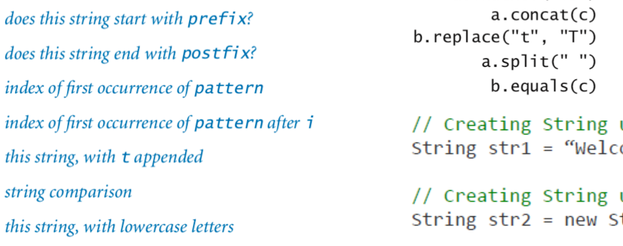
prints the following output:

```
1
2
orange
apple
```

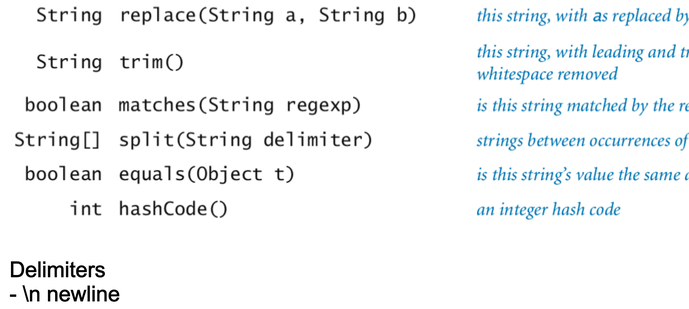
## Instance variables




## Instance methods




## Constructors




## Commenting

- // single line comment
- /* multi-line comment*/
- /**
  * Documentation (Javadoc)
  */

## Access Control

- public (accessible wherever the class is visible)
- protected (accessible only to the class and its subclasses)
  - Also accessible to all classes in same package
- private (hidden from all other classes)
- if none of the above are stated --> package visibility
  - visible to all classes in same package
  - if no package specified, puts into "default" package

## String datatype

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* |
| 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* |

```
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 |

```
// Creating String using literal
String str1 = "Welcome";

// Creating String using new keyword
String str2 = new String("Edureka");

str1==str2 //compare the address;
String newStr = str1.equals(str2); //compares the values
String newStr = str1.equalsIgnoreCase() //
newStr = str1.length() //calculates length
newStr = str1.charAt(i) //extract i'th character
newStr = str1.toUpperCase() //returns string in ALL CAPS
newStr = str1.toLowerCase() //returns string in ALL LOWERCASE
newStr = str1.replace(oldVal, newVal) //search and replace
newStr = str1.trim() //trims surrounding whitespace
newStr = str1.contains("value"); //Check for the values
newStr = str1.toCharArray(); //Convert into character array
newStr = str1.IsEmpty(); //Check for empty String
newStr = str1.endsWith(); //Checks if string ends with the given suffix
```

## Good Practice

- Naming convention
  - Use camel case for variables (no underscores): myVaraible
  - Use uppercase Pascal case for class names: MyClass
  - All caps for constants: MY_CONSTANT
  - Provide meaningful names
- Access control
  - All variables must be private
  - Instance methods must be public (sometimes protected)
  - Helper methods must be private
- Programming style
  - Keep lines short (many terminals/tools have trouble handling lines longer than 80 characters)
  - Variables should be declared in the innermost block that encloses all its uses
    - If a variable only used inside of if statement, declare it inside the if statement
  - Always use braces for selection/repetition statements even if there is only one thing to do
    - The only exception is if the whole statement fits on one line
- Class Design
  - Write short methods (5 lines or less goal)
  - Use few instance variables in a class (5 or less goal)
  - Public instance method should never call each other (within the same class)
    - Use private helper methods instead
  - Helper methods can be shared using a public wrapper method
- OOP Goals
  - Increase cohesion (Classes should consist of few well defined functions, not many general ones)
  - Decrease coupling (Minimize interaction between classes)

## Delimiters

- \n newline
- \t: tab
- \r carriage return
  - used in place of \n for some OS

String.split("\\s+") --> Groups all white space as delimiter
- may need to trim() first

## ArrayList (variable length)

- ArrayList<type> name = new ArrayList<type>();
  - new ArrayList<type>(arrayList);
- ArrayList methods
  - arrList.set(index, value)
  - arrList.get(index)
- Using iterator

## Iterator

- Methods
  - hasNext()
  - next() // returns the element
  - remove() removes the last element

```
boolean add(Object)              // Adds a new object to the list
boolean addAll(Collection)       // Adds a collection to the list
void clear()                     // clears the list
boolean contains(Object)         // returns true if list contains an specific object
boolean equals(Object)           // Compares the specified object with this collection
boolean isEmpty()                // returns true if list is empty
Iterator iterator()              // returns an iterator over the elements of the list
boolean remove(Object)           // removes and specific element
boolean removeAll(Collection)    // removes all
int size()                       // returns the number of elements
Object[] toArray()               // returns an array containing all elements
```

Important notes:
- When comparing strings, use s1.equals(s2) and not ==
- Loops consist of 3 expressions
  - Initialization expression
  - Test expression
  - Updating expression
- Java is called a framework because class object is inherited/extended?
- Java is called a framework because it is never finished (always updated)

## Interfaces

- Syntax: `public class Point implements Color`
- Unimplemented declarations of methods and/or constants
- All methods inside of an interface are abstract (either explicitly or implicitly)
- Classes can implement more than one interface (pseudo multiple inheritance)
- Interfaces can extend more than one interface (multiple inheritance)
- A reference to an interface can refer to instances of any classes that implement that interface
  - i.e. A function that takes an interface as an argument can accept any object that inherits from that interface as an argument

## Abstract Classes vs Interfaces

- Abstract classes can only extend one class
  - Interfaces can extend any number of interfaces (but only interfaces)
- Abstract classes can have both abstract and concrete methods
  - Interfaces can only have abstract methods (unless method declared default)
- Abstract classes must use the keyword abstract to have abstract methods
  - Interfaces optionally use the keyword abstract (methods may be implicitly abstract)
- Abstract classes can have protected and public abstract methods
  - Interfaces can only have public abstract methods
- Abstract classes can have static/non-static, final, or static final variables with any access identifier
  - Interfaces can only have final (constant variables)

## When to use abstract classes vs interfaces

- If no non-constant data (instance variables) are to be inherited, they are functionally the same
- Abstract classes should be used when (non-constant) instance variables are to be inherited
  - Interfaces can't have non-constant variables
- Interfaces should be used when only constants and/or methods are to be inherited

## SOLID Principles

- Single Responsibility Principle
  - A class should only have one responsibility
  - Increases cohesion, decreases coupling, easier to test, better organization
  - If any of the OLID principles are violated, the S principle is violated
- Open Closed Principle
  - Classes should be open for extension and closed for optimization
  - Instead of changing a working class, inherit and make changes
  - Prevents working code from being modified and introducing bugs
- Liskov Substitution Principle
  - A child class should contain at least everything that its parent has
  - If we substituted the child class for the parent class, program behavior would not be disrupted
  - Violated if parent classes are not general enough for their children
- Interface Segregation Principle
  - Larger interfaces should be split into smaller ones
  - Ensures that implementing classes are only receiving the methods that they need
- Dependency Inversion Principle
  - Classes should depend on abstraction
  - Solved using the dependency injection design pattern
  - By having abstract classes as arguments in methods (dependency), children of the abstract class can be used as arguments in the method.
  - This allows changes in arguments to be made without having to change the dependent code

## Object Models (Four main elements)

- Abstraction (Implementation hiding)
  - Represent complex concept with simplified models
- Encapsulation (Information hiding)
  - Public interfaces with hidden implementation*********
- Hierarchy (2 hierarchies)
  - HAS-A (Object structure AKA assocation, aggregation, composition)
  - IS-A (Specializations of general objects AKA inheritance)
    - Also consider use-a (dependency/association)
- Modularity (Can work on one isolated aspect of big picture)

## Class relationships in code

- Association
  - Classes contain instance variables of each other
  - 1 way association
    - Class A has an instance variable of Class B but not the other way around
  - Reflexive assocation
    - Class A has an instance variable of itself
- Aggregation
  - Class A has an instance variable of Class B but not the other way around
  - AND Class B instance variable is initialized in the constructor of Class A
- Composition
  - Class A has an instance variable of Class B but not the other way around
  - AND Class A also constructs an object of type B
- Dependency
  - Class A does not have an instance variable of Class B
  - Class A receives an object of Class B as an argument in one of its methods
  - OR Class A uses an object of Class B as a local variable?
- Inheritance
  - Class B extends Class A in its definition
- Implementation?
  - Class A implements Interface A

## Inheritance

- Syntax: `class ThreeDPoint extends Point {`
- Protected variables in parent can be seen by children
- Private variables in parent can be called by referring to super (i.e. super.setX())
- A subclass can invoke a method of any super calss as long as it has not been redefined
- super() MUST be called in the first line of the constructor of the subclass

## Polymorphism

- When a function is tailored to work with more than one object type
- When a parent function is redefined in a child classes so method works with both objects
- Works with inheritance to allow functions to be compatible with multiple object types
- Diamond problem fix: ParentClass.super().method()

## Abstract and concrete classes

- Abstract classes cannot be instantiated (concrete classes can)
- Abstract methods do not contain bodies (concrete methods have bodies)
  - Subclasses are expected to define body of abstract methods

## Final classes and methods

- Final classes cannot be inherited from (extended)
- Therefore, final methods cannot be redefined
- Final classes and methods can be optimized by the compiler

Cloning
- If you set a = b, then variable a and b now point ot the same space in memory (aliasing, shallow copy)
  - This means, if you change a, b will also change
- If you want to make an independent (deep) copy of something, you have to clone it.
- Steps to clone:
  - If cloning a user defined class, implement "Cloneable" interface into class
  - Redefine the clone() method by adding clone statement for object and user defined instance variables

```
public Object clone() throws CloneNotSupportedException
{
    Shape temp = (Shape)super.clone();
    temp.origin = (Point)origin.clone();
    return temp;
}
```

  - Note that, user defined instance variables must be copied using another clone statement.
    - Therefore, class Point must also implement "Cloneable" and redefine define its clone() method.

```
public Object clone() throws CloneNotSupportedException
{
    Point temp = (Point)super.clone();
    return temp;
}
```

  - If the class only contains Java's built-in data types, only the object clone statement is necessary.
    - This can be seen when cloning object of type Point
  - Call the clone method in the main function

```
try
{
    rectangleB = (Rectangle)rectangleA.clone();
}
catch(CloneNotSupportedException e)
{
    System.out.println("Can't clone rectangle A");
}
```

If cloning array of user defined objects, have to loop through and clone each instance

Exception Handling
- 3 kinds of exceptions
  - Checked exceptions (Subject to catch or specify requirement)
  - Error (Exceptional conditions external to the application)
  - Runtime exceptions (Exceptional conditions internal to application)
- An exception is caught by one of the following
  - A surrounding block of code
  - Some calling code
  - The JVM
- Exception handling advantages
  - Separating error-handling code from "regular" code
  - Propagating errors up the call stack
  - Grouping and differentiation of error types
- Generally, front and back end should each do their own exception handling
- Can define your own exception class by extending "Exception" into class
- The "finally" clause is always executed after code in the try/catch blocks
- e.getStrackTrace() <-- useful
- Order exceptions from most specific to least specific

Packages
- Package coupling tips
  - Packages should not be cross coupled
  - Packages in lower layers should not be dependent upon packages in higher layers
  - In general, dependencies should not skip layers
- Accessing package contents
  - Use fully qualified name (i.e. Geometry.Point myPoint;)
  - Import part of all of a package and use a simple name (i.e. import Geometry*)

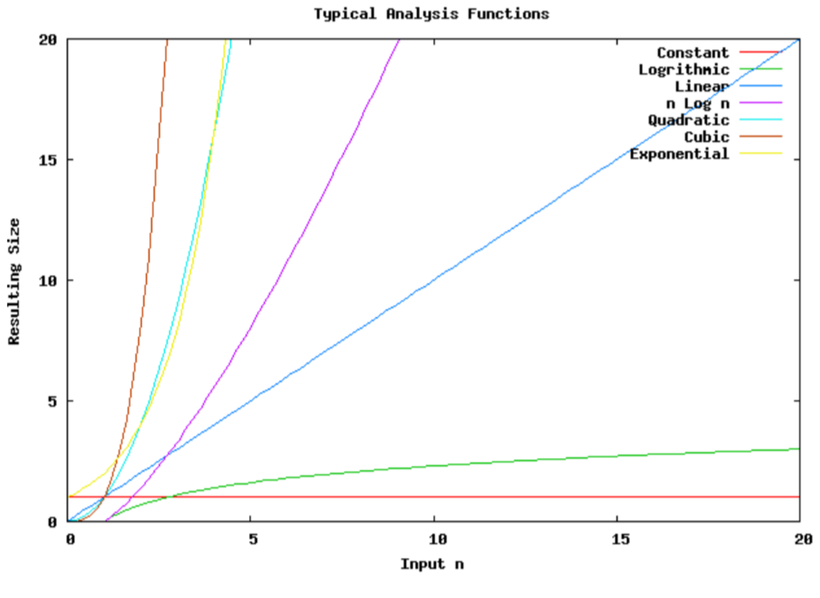
GUI
- How events are handled
  - A listener object is an instance of a class that implements an interface called a listener interface
  - An event source must be registered to a listener object
  - The event source sends out event object to all registered listeners when an event happens
  - The listener objects will use the information events encapsulated in the event to determine their reaction to the event
- Methods of implementing event listener
  - Listener class (Organized but results in a lot of listener class scripts)
  - Named inner classes (Everything in the same class but the class can become very large)
  - Anonymous inner class (Obsolete due to lambda, uses anonymous object to perform function rather than storing
  - Lambda functions (Functional programming)

MVC
- Side note
  - Tier (physical separation, different machines/virtual machines)
  - Layer (logical separation, different script/package)
- MVC Roles
  - Model (manipulates and stores data)
    - Contains logic/methods for manipulating information
    - Stores information in instance variables
  - View (receives and presents data)
    - Presents information to user in GUI
    - Accepts input from user
  - Controller (connects model to controller)
    - Relays information between view and model in logical way
- Benefits
  - Can modify view without impacting the underlying model
- Problems
  - Need to maintain consistency in the views
  - Need to update multiple views of the common data model (or subject)
  - Need clear, separate responsibilities for presentation, interaction, computation, and persistence

Database
- SQL injection (one of the most common hacking techniques)
  - Uses dynamic queries to concatenate malicious code into the query
  - Solved by using prepared statements
- ONLY USE PREPARED STATEMENTS

Complexity
- A measure of the difficulty of performing a computation in terms of:
  - The time required (time complexity)
  - The number of steps/arithmetic operations required (computational complexity)
  - The amount of memory required (space complexity)
- Tells us how the time/space required to solve a problem varies with input data size
- Can't analyze complexity empirically very effectively
- To get around this, we analyze the algorithm directly in terms of
  - The number of primitive operations executed (assuming each takes up about the same amount of time)
    - Results in a function t(n)
- O(n lg n) and above are considered efficient for high n
  - The others are only fit for small input quantities
- O notation specifies an upper bound on a function
- Ω notation specifies a lower bound on a function
- Θ notation specifies upper and lower bounds simultaneously

- The types of algorithms are:
  - O(1) constant
  - O(lg *n*) logarithmic
  - O(*n*) linear
  - O(*n* lg *n*) N-Log-N
  - O($n^2$) quadratic
  - O($n^3$) cubic
  - O($2^n$) exponential




Sequential search
- Iterate through list, comparing each element to the query key until a match is found or the end of the list is reached.
- Best case: O(1), match found immediately
- Worst case: O(n), match at end or no match found
- Average O(n), (n+1)/2 comparisons performed

Binary search (needs sorted list)
- Divide array in half by locating middle item and comparing it to query key
  - If key < middle item, repeat for left half of array
  - If key > middle item, repeat for right half of array
  - If key = middle item, return
- Best case: O(1)
- Worst case O(lg n)
- Average case O(lg n), in practice about 2x faster than worst case

Interpolation search
- Binary search but midpoint is set to where the item is likely to occur
- Best case: O(1)
- Worst case O(lg n)
- Average case O(lg n), in practice little bit faster than binary search

Sorting
- Sort types
  - Internal sort (data kept in primary storage in RAM using arrays)
  - External sort (data kept in secondary storage on disk or tape
  - In-place sort (sort exchanges items in place, requires no extra memory)
  - Stable sort (sort that preserves the relative order of equal keys)
    - E.g. Sort a list of people first my name then by age (Now age groups are in alphabetical order)
- Typically analyzed using Big O notation for:
  - Best case (already sorted)
  - Worst case (data in reverse order)
  - Random case (data in random order)
- Simple sorting algorithms
  - Bubble, selection, and insertion sort
- Complex sorting algorithms
  - Merge sort and quick sort
- The theoretical best running time we can achieve for a comparison-based sort is O(n lg n)

Bubble sort
- Compare successive pairs of items, swapping them if out of order
- O(n^2) for all cases (unless optimized)

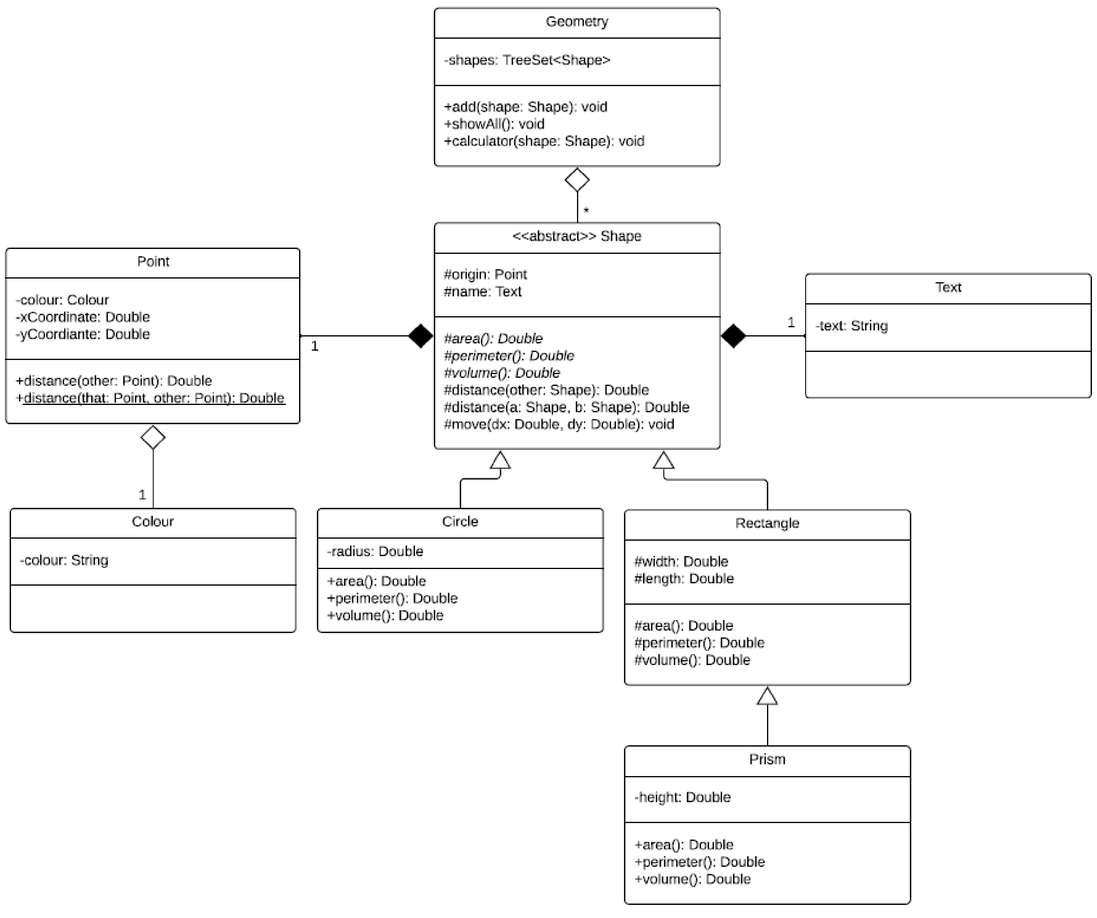
Selection sort
- Takes smallest item of list and swaps it to front
- Repeats process for each item of the array until second last item
- After each pass, the low part of the array is sorted and is no longer considered

Insertion sort
- Examines each item of the array and inserts it into new array in ascending order, then overwrites old array
  - Can be done in place such that an ordered array emerges on left side

Merge sort (needs extra memory)
- Divide array into two equal-size sub-arrays
- Sort each sub-array (call merge sort recursively)
- Merge the sub-arrays into a temporary array
- Copy the temporary array back into the original array

Quick sort (in place sort)
- Choose one element to be the pivot
- Partition the array into 2 subarrays, such that:
  - Subarray1 contains only elements <= pivot
  - Pivot is in its final position in the array
  - Subarray2 contains only elements >= pivot
- Apply recursively until subarray length <= 1
- WORST CASE MAY BE O(n^2) ACCORDING TO NOTES

| Sorting Algorithms | Space complexity | Time complexity | | |
|---|---|---|---|---|
| | Worst case | Best case | Average case | Worst case |
| Insertion Sort | O(1) | O(n) | O($n^2$) | O($n^2$) |
| Selection Sort | O(1) | O($n^2$) | O($n^2$) | O($n^2$) |
| Bubble Sort | O(1) | O(n) | O($n^2$) | O($n^2$) |
| Mergesort | O(n) | O(n log n) | O(n log n) | O(n log n) |
| Quicksort | O(log n) | O(n log n) | O(n log n) | O(n log n) |
| Heapsort | O(1) | O(n log n) | O(n log n) | O(n log n) |




UML Notation
- Member access modifiers
  - private (-)
  - public (+)
  - protected (#)
  - package (~)
  - derived (/)
  - static (underlined)
  - abstract (italicized)
- Relationships
  - Association (solid line)
  - Aggregation (white diamond, solid line)
  - Composition (black diamond, solid line)
  - Inheritance/extension (triangle, solid line)
  - Implementation (triangle, dotted line)
  - Dependency (v-head, dotted line)