```
001 // A Single line comment
002
003 /* A
004  * Multiple line
005  * comment
006  */
007
008 // You can import libraries with helpful methods using import
009
010 import java.util.Scanner;
011 import java.util.*;
012
013 // A class defines the attributes (fields) and capabilities
    (methods) of a real world object
014
015 public class Animal {
016
017     // static means this number is shared by all objects of type
    Animal
018     // final means that this value can't be changed
019     public static final double FAVNUMBER = 1.6180;
020
021     // Variables (Fields) start with a letter, underscore or $
022     // Private fields can only be accessed by other methods in the
    class
023
024     // Strings are objects that hold a series of characters
025     private String name;
026
027     // An integer can hold values from -2 ^ 31 to (2 ^ 31) -1
028     private int weight;
029
030     // Booleans have a value of true or false
031     private boolean hasOwner = false;
032
033     // Bytes can hold the values between -128 to 127
034     private byte age;
035
036     // Longs can hold the values between -2 ^ 63 to (2 ^ 63) - 1
037     private long uniqueID;
038
039     // Chars are unsigned ints that represent UTF-16 codes from 0
    to 65,535
040     private char favoriteChar;
041
042     // Doubles are 64 bit IEEE 754 floating points with decimal
    values
043     private double speed;
044
045
```

```
        // Floats are 32 bit IEEE 754 floating points with decimal
    values
046     private float height;
047
048     // Static variables have the same value for every object
049     // Any variable or function that doesn't make sense for an
    object to have should be made static
050     // protected means that this value can only be accessed by
    other code in the same package
051     // or by subclasses in other packages
052
053     protected static int numberOfAnimals = 0;
054
055     // A Scanner object allows you to except user input from the
    keyboard
056     static Scanner userInput = new Scanner(System.in);
057
058     // Any time an Animal object is created this function called
    the constructor is called
059     // to initialize the object
060     public Animal(){
061
062         // Shorthand for numberOfAnimals = numberOfAnimals + 1;
063         numberOfAnimals++;
064
065         int sumOfNumbers = 5 + 1;
066         System.out.println("5 + 1 = " + sumOfNumbers);
067
068         int diffOfNumbers = 5 - 1;
069         System.out.println("5 - 1 = " + diffOfNumbers);
070
071         int multOfNumbers = 5 * 1;
072         System.out.println("5 * 1 = " + multOfNumbers);
073
074         int divOfNumbers = 5 / 1;
075         System.out.println("5 / 1 = " + divOfNumbers);
076
077         int modOfNumbers = 5 % 3;
078         System.out.println("5 % 3 = " + modOfNumbers);
079
080         // print is used to print to the screen, but it doesn't end
    with a newline \n
081         System.out.print("Enter the name: \n");
082
083         // The if statement performs the actions between the { } if
    the condition is true
084         // userInput.hasNextLine() returns true if a String was
    entered in the keyboard
085         if(userInput.hasNextLine()){
086
087
```

```
                // this provides you with a way to refer to the object
    itself
088             // userInput.nextLine() returns the value that was
    entered at the keyboard
089             this.setName(userInput.nextLine());
090
091             // hasNextInt, hasNextFloat, hasNextDouble,
    hasNextBoolean, hasNextByte,
092             // hasNextLong, nextInt, nextDouble, nextFloat,
    nextBoolean, etc.
093
094         }
095
096         this.setFavoriteChar();
097         this.setUniqueID();
098
099     }
100
101     // It is good to use getter and setter methods so that you can
    protect your data
102     // In Eclipse Right Click -> Source -> Generate Getter and
    Setters
103
104     public String getName() {
105         return name;
106     }
107
108     public void setName(String name) {
109         this.name = name;
110     }
111
112     public int getWeight() {
113         return weight;
114     }
115
116     public void setWeight(int weight) {
117         this.weight = weight;
118     }
119
120     public boolean isHasOwner() {
121         return hasOwner;
122     }
123
124     public void setHasOwner(boolean hasOwner) {
125         this.hasOwner = hasOwner;
126     }
127
128     public byte getAge() {
129         return age;
130     }
131
```

```
132     public void setAge(byte age) {
133         this.age = age;
134     }
135 
136     public long getUniqueID() {
137         return uniqueID;
138     }
139 
140     // Method overloading allows you to accept different input with
    the same method name
141     public void setUniqueID(long uniqueID) {
142         this.uniqueID = uniqueID;
143 
144         System.out.println("Unique ID set to: " + this.uniqueID);
145     }
146 
147     public void setUniqueID() {
148 
149         long minNumber = 1;
150         long maxNumber = 1000000;
151 
152         // Generates a random number between 1 and 1000000
153         this.uniqueID = minNumber + (long)(Math.random() *
    ((maxNumber - minNumber) + 1));
154 
155         // You can cast from one primitive value into another by
    putting what you want between ( )
156         // (byte) (short) (long) (double)
157         // (float), (boolean) & (char) don't work.
158         // (char) stays as a number instead of a character
159 
160         // You convert from a primitive to a string like this
161         String stringNumber = Long.toString(maxNumber);
162 
163         // Byte.toString(bigByte); Short.toString(bigShort);
    Integer.toString(bigInt);
164         // Float.toString(bigFloat); Double.toString(bigDouble);
    Boolean.toString(trueOrFalse);
165 
166         // You convert from a String to a primitive like this
167         int numberString = Integer.parseInt(stringNumber);
168 
169         // parseShort, parseLong, parseByte, parseFloat,
    parseDouble, parseBoolean
170 
171         System.out.println("Unique ID set to: " + this.uniqueID);
172     }
173 
174     public char getFavoriteChar() {
175         return favoriteChar;
176     }
```

```
177
178     public void setFavoriteChar(char favoriteChar) {
179         this.favoriteChar = favoriteChar;
180     }
181
182     public void setFavoriteChar() {
183
184         int randomNumber = (int) (Math.random() * 126) + 1;
185
186         this.favoriteChar = (char) randomNumber;
187
188         // if then else statement
189         // > < == != >= <=
190         if(randomNumber == 32){
191
192             System.out.println("Favorite character set to: Space");
193
194         } else if(randomNumber == 10){
195
196             System.out.println("Favorite character set to: New
    Line");
197
198         } else {
199
200             System.out.println("Favorite character set to: " +
    this.favoriteChar);
201
202         }
203
204         // Logical operators
205         // ! : Converts the boolean value to its right to its
    opposite form ie. true to false
206         // & : Returns true if boolean value on the right and left
    are both true (Always evaluates both boolean values)
207         // && : Returns true if boolean value on the right and left
    are both true (Stops evaluating after first false)
208         // | : Returns true if either boolean value on the right or
    left are true (Always evaluates both boolean values)
209         // || : Returns true if either boolean value on the right
    or left are true (Stops evaluating after first true)
210         // ^ : Returns true if there is 1 true and 1 false boolean
    value on the right or left
211
212         if((randomNumber > 97) && (randomNumber < 122)){
213
214             System.out.println("Favorite character is a lowercase
    letter");
215
216         }
217
218
```

```
        if(((randomNumber > 97) && (randomNumber < 122)) ||
((randomNumber > 64) && (randomNumber < 91))){
219
220             System.out.println("Favorite character is a letter");
221
222         }
223
224         if(!false){
225
226             System.out.println("I turned false to " + !false);
227
228         }
229
230         // The ternary operator assigns one or another value based
on a condition
231         int whichIsBigger = (50 > randomNumber) ? 50 :
randomNumber;
232
233         System.out.println("The biggest number is " +
whichIsBigger);
234
235         // The switch statement is great for when you have a
limited number of values
236         // and the values are int, byte, or char unless you have
Java 7 which allows Strings
237         switch(randomNumber){
238
239         case 8 :
240             System.out.println("Favorite character set to:
Backspace");
241             break;
242
243         case 9 :
244             System.out.println("Favorite character set to:
Horizontal Tab");
245             break;
246
247         case 10 :
248         case 11 :
249         case 12 :
250             System.out.println("Favorite character set to:
Something else weird");
251             break;
252
253         default :
254             System.out.println("Favorite character set to: " +
this.favoriteChar);
255             break;
256
257         }
258
```

```
259     }
260
261     public double getSpeed() {
262         return speed;
263     }
264
265     public void setSpeed(double speed) {
266         this.speed = speed;
267     }
268
269     public float getHeight() {
270         return height;
271     }
272
273     public void setHeight(float height) {
274         this.height = height;
275     }
276
277     protected static int getNumberOfAnimals() {
278         return numberOfAnimals;
279     }
280
281     // Since numberOfAnimals is Static you must set the value using
    the class name
282     public void setNumberOfAnimals(int numberOfAnimals) {
283         Animal.numberOfAnimals = numberOfAnimals;
284     }
285
286     protected static void countTo(int startingNumber){
287
288         for(int i = startingNumber; i <= 100; i++){
289
290             // continue is used to skip 1 iteration of the loop
291             if(i == 90) continue;
292
293             System.out.println(i);
294
295         }
296
297     }
298
299     protected static String printNumbers(int maxNumbers){
300
301         int i = 1;
302         while(i < (maxNumbers / 2)){
303
304             System.out.println(i);
305             i++;
306
307             // This isn't needed, but if you want to jump out of a
    loop use break
```

```
308             if(i == (maxNumbers/2)) break;
309 
310         }
311 
312         Animal.countTo(maxNumbers/2);
313 
314         // You can return a value like this
315         return "End of printNumbers()";
316 
317     }
318 
319     protected static void guessMyNumber(){
320 
321         int number;
322 
323         // Do while loops are used when you want to execute the
    code in the braces at least once
324         do {
325 
326             System.out.println("Guess my number up to 100");
327 
328             // If what they entered isn't a number send a warning
329             while(!userInput.hasNextInt()){
330 
331                 String numberEntered = userInput.next();
332                 System.out.printf("%s is not a number\n",
    numberEntered);
333 
334             }
335             number = userInput.nextInt();
336 
337         }while(number != 50);
338 
339         System.out.println("Yes the number was 50");
340 
341     }
342 
343     // This will be used to demonstrate polymorphism
344     public String makeSound(){
345 
346         return "Grrrr";
347 
348     }
349 
350     // With polymorphism we can refer to any Animal and yet use
    overridden methods
351     // in the specific animal type
352     public static void speakAnimal(Animal randAnimal){
353 
354         System.out.println("Animal says " + randAnimal.makeSound
    ());
```

```
355
356     }
357
358     // public allows other classes to use this method
359     // static means that only a class can call for this to execute
360     // void means it doesn't return a value when it finishes
    executing
361     // This method can except Strings that can be stored in the
    String array args when it is executed
362
363     public static void main(String[] args){
364
365         Animal theDog = new Animal();
366
367         System.out.println("The animal is named " + theDog.getName
    ());
368
369         System.out.println(Animal.printNumbers(100));
370
371         Animal.countTo(100);
372
373         Animal.guessMyNumber();
374
375         // An array is a fixed series of boxes that contain
    multiple values of the same data type
376         // How you create arrays
377         // int[] favoriteNumbers;
378         // favoriteNumbers = new int[20];
379
380         int[] favoriteNumbers = new int[20];
381
382         favoriteNumbers[0] = 100;
383
384         String[] stringArray = {"Random", "Words", "Here"};
385
386         // for(dataType[] varForRow : arrayName)
387         for(String word : stringArray)
388         {
389
390             System.out.println(word);
391
392         }
393
394         // This is a multidimensional array
395         String[][][] arrayName = { { { "000" }, { "100" },
    { "200" }, { "300" } },
396                 { { "010" }, { "110" }, { "210" }, { "310" } },
397                 { { "020" }, { "120" }, { "220" }, { "320" } }};
398
399         for(int i = 0; i < arrayName.length; i++)
400         {
```

```
401             for(int j = 0; j < arrayName[i].length; j++)
402             {
403 
404                 for(int k = 0; k < arrayName[i][j].length; k++)
405                 {
406                     System.out.print("| " + arrayName[i][j][k] + "
    ");
407 
408                 }
409             }
410 
411             System.out.println("|");
412 
413         }
414 
415         // You can copy an array (stringToCopy, indexes to copy)
416         String[] cloneOfArray = Arrays.copyOf(stringArray, 3);
417 
418         // You can print out the whole array
419         System.out.println(Arrays.toString(cloneOfArray));
420 
421         // Returns the index or a negative number
422         System.out.println(Arrays.binarySearch(cloneOfArray,
    "Random"));
423 
424     }
425 
426 }
```