Arrays

[1] This is the Java expression that denotes the number of elements in the array A.

[2] This is one advantage of representing a sequence of elements as an array rather than as a list.

[3] Suppose that B is an array of booleans. This is a sequence of statements that swaps the contents of the first and last slots of B.

[4] This is a definition of a `concatAll()` method that concatenates all of the elements of an array of strings into a single string. For example, suppose `a` is defined as follows:

```java
String [] a = {"ab","cde","","f","ghij"};
```

Then `concatAll(a)` returns the string "abcdefghij".

[5] This is a definition of a class method satisfying the following contract:

```java
public IntList intArrayToList (int [] A);
Returns an IntList containing all of the elements of A in the same order.
```
Objects

[1] A class declaration typically includes these entities, used to keep track of an object’s state.

[2] This keyword is used to signify a variable or method that is not tied to a specific instance of a class.

[3] This is a list of all the different kinds of (1) methods and (2) variables that can be in a Java class declaration.

[4] This is displayed in the Java Console window by an animation that contains a single sprite create via new TextSprite(2,1), where the TextSprite class is defined as follows:

```java
public class TextSprite extends Sprite {
    private int x = 17;
    public TextSprite (int a, int b) {x = 10*a + b;}
    public void updateState() {x = x/2 - 1;}
    public void drawState() {
        if (x > 0) System.out.println(2*x);
    }
}
```

[5] This is displayed in the Java Console window when the main method of the following Counter class is executed:

```java
public class Counter {
    private static int c = 0;
    private int i;
    public Counter () {c = c + 1; i = 0;}
    public int print () {
        i = i + 1;
        System.out.println("c = " + c + "; i = " + i);
    }
    public void main (String [] args) {
        Counter a = new Counter(); a.print(); a.print();
        Counter b = new Counter(); b.print(); a.print();
    }
}
```
Worlds

[1] Buggles love to eat these.

[2] Suppose that $w$ is a PictureWorld picture of the wedge shown below in Figure 1. This is a PictureWorld expression that denotes the picture in Figure 2.

![Figure 1](image1.png) ![Figure 2](image2.png)

[3] This is the picture drawn in an applet by the following statements. (Indicate relevant coordinates in your picture.)

```java
Graphics g = getGraphics();
Point p1 = new Point(10, 20);
Point p2 = new Point(30, 60);
g.setColor(Color.red);
g.drawOval(p1.x, p1.y, p1.y, p1.y);
g.drawRect(p1.x, p1.y, p2.x, p2.y);
Polygon p = new Polygon():
p.addPoint(p1.x, p1.y);
p.addPoint(p2.x, p2.y);
p.addPoint(p1.x, p2.y);
g.setColor(Color.green);
g.fillPoly();
```

[4] This is a definition of the buggle method satisfying the following contract:

```java
public boolean canGoForwardBy (int n);
```

Returns true if the buggle would not encounter a wall in forward(n), and false otherwise. Executing this method should leave the state of the buggle unchanged.
This is the picture drawn by invoking the turtle method `pattern(40)` on a new turtle, where `pattern` is defined as follows:

```java
public void pattern (int n) {
    if (n < 10) {
        fd(n)
    } else {
        pattern(n/2);
        lt();
        fd(n);
        bd(n);
        rt();
        pattern(n/2);}}
```
Lists

[1] When defining a recursive list method, a good strategy is to assume you can successfully invoke the method on this part of the list.

[2] This is one advantage of storing a sequence of elements in a list as opposed to an array.

[3] This list is the result of applying the following mystery() method to the list [2, 3, 9, 5, 6, 4]

```java
public IntList mystery (IntList L){
    if(isEmpty(L)){
        return empty();
    } else if ((head(L) % 3) == 0) {
        return mystery(tail(L));
    } else {
        return prepend(2*head(L),
                        mystery(tail(L)));
    }
}
```

[4] How many new list nodes are created by the invocation appendages(ns), where ns is the list [1,2,3], and appendages is defined below:

```java
public IntList appendages (IntList L) {
    if(isEmpty(L)) {
        return L;
    } else {
        return append(L, appendages(tail(L)));
    }
}
```

```java
public IntList append (IntList L1, IntList L2) {
    if(isEmpty(L1)) {
        return L2;
    } else {
        return prepend(head(L1), append(tail(L1), L2));
    }
}
```

[5] This is the definition of a method doubles that takes an IntList L as its single argument and returns an IntListList whose list elements are the the result of doubling all integers in the successive tails of L. For example:

```
doubles(IL.fromString("[7,2,3]"))
```

returns the following list of lists:

```
[[14, 4, 6], [4, 6], [6], []]
```

Use IL. and ILL. appropriately.
Bugs That Bite

[1] This is a bug in the following array method.

```java
public static int product (int [] a) {
    int result = 1;
    for (int i = 0; i <= a.length; i++) {
        result = a[i] * result;
    }
    return result;
}
```

[2] This is a bug in the following turtle method:

```java
public int spiral (int n) {
    if (n == 0) {
        return 0;
    } else {
        fd(n); lt();
        spiral(n/2);
        rt(); bd(n);
    }
}
```

[3] This is a bug in the following method to determine if an integer list is sorted:

```java
public static boolean isSorted (IntList L) {
    if (isEmpty(L)) {
        return true;
    } else {
        return (head(L) <= head(tail(L)))
                && isSorted(tail(L));
    }
}
```
[4] These are two bugs in the following class declaration:

```java
public class Circle {

    private Point center;
    private int radius;

    public Circle (Point c, int radius) {
        Point center = c;
        radius = radius;
    }

    public void draw (Graphics g) {
        g.drawOval(center.x - radius, center.y - radius,
                    2*radius, 2*radius);
    }
}
```

[5] These are two bugs in the following isMember method for determining if a given integer is in an array of integers sorted from low to high:

```java
// Assume a is sorted from low to high
public static boolean isMember (int n, int[] a) {
    int i = a.length - 1;
    while ((n > a[i]) && (i >= 0)) {
        i--;
    }
    return (i >= 0);
}
```
Potpourri

[1] In the Java Execution Model, this is created when an instance method is invoked.

[2] This special type of recursion can also be written as a while loop.


[4] This is a list of all of the following that are Java expressions (as opposed to statements). (Note: all semicolons have been omitted so they don’t provide a cue.)

(a) 1 + 2

(b) n == 0

(c) x = 0

(d) debby.forward(7)

(e) ellie.isOverBagel()

(f) if (x > 0) {return x} else {return -x}

[5] This is a definition of the buggle method satisfying the following contract:

```java
public void forwardTurningLeft(int n);
Moves the buggle forward a total of n spaces, turning left whenever the buggle encounters a wall. (Turning does not count as “moving forward a space”.)
```