AD TESTING SERVICES
2013-2014 UIL COMPUTER SCIENCE PRACTICE TEST 1

UIL COMPUTER SCIENCE PRACTICE TESTS FOR 2013-2014

QUESTION 1

What is AB_{16} plus CD_{16}?

A. $ABCD_{16}$ B. $E18_{16}$ C. 91_{16} D. 324_{16} E. 178_{16}

QUESTION 2

What is output by the code to the right?

A. p B. 28 C. 76

D. 1 E. n / 3 + 6

```
int m = 64;
int n = m + 8 / 4;
int p = n / 3 + 6;
System.out.println(p);
```

QUESTION 3

What is output by the code to the right?

A. total B. -55 C. 101

D. -10 E. 11

```
int total = 0;
for (int i = 10; i > -1; i--) {
  total -= i;
}
System.out.print(total);
```

QUESTION 4

What is output by the code to the right?

A. oI B. so

C. kC D. c!

E. There is no output due to a syntax error.

```
String c1 = "CS Rocks!";
String c2 = "UIL is Cool!";
String c3 = c1.charAt(7);
String c4 = c2.charAt(9);
System.out.print(c3 + c4);
```

QUESTION 5

What is output by the code to the right?

A. true 0 B. true 3

C. false 3 D. false null

E. null null

```
boolean[] st = new boolean[3];
System.out.print(st[0] || st[1] || st[2]);
System.out.print(" " + st.length);
```

QUESTION 6

What is output by the code to the right?

A. 0.5 B. 5.5

C. 1.1 D. 1.0

E. 0.0

```
double db = 5.5;
System.out.print(db%5%5);
```

QUESTION 7

Which answer is logically equivalent to the following boolean expression, where p, q, and r are int variables?
`!((p < q) && (r < q))`

A. `(p >= q) || (q >= r)` B. `(p >= q) && (q >=r)` C. `!(p < q) && !(r < q)`

D. `!(q >= p) || !(r < q)` E. `!(p == r)`

QUESTION 8

What is output by the code to the right?

A. `yes` B. `no`

C. `yesno` D. `nono`

E. `yesnono`

```
double one = 1.11;
if(one / 2 > 0)
  System.out.print("yes");
  System.out.print("no");
if(one / 2 > 0.5)
System.out.print("no");
```

QUESTION 9

What replaces **<*1>** in the code to the right to give the number of characters in `String s`?

A. `s.length()` B. `s.len`

C. `s.numChars()` D. `s.charCount`

E. `s.size`

QUESTION 10

Assume **<*1>** is filled in correctly. What is returned by `dowhat("abc123")`?

A. `-6` B. `4` C. `3`

D. `6` E. `-5`

```
public static int dowhat(String s) {
  int total = 0;
  for (int i=1; i < <*1>; i++) {
    char curr = s.charAt(i);
    char prev = s.charAt(i-1);
    if (curr >= prev)
      total++;
  }
  return total;
}
```

QUESTION 11

What is output by the code to the right?

A. `-2` B. `3` C. `2`

D. `-5` E. `0`

```
System.out.print(Math.min(-5, 3));
```

QUESTION 12

What is output by the code to the right?

A. `[0, 8, 0]` B. `[15, 21, 6]`

C. `[3, 7, 0]` D. `[10, 13, 6]`

E. There is no output due to a syntax error.

```
int[] arr1 = {5, 8, 0};
int[] arr2 = {10, 13, 6};
System.out.print(arr1 & arr2);
```

QUESTION 13

How many lines are output by the code to the right?

A. 1

B. 2

C. 3

D. 4

E. There is no output due to a syntax error.

```
System.out.print("o\ne");
System.out.println("two");
System.out.print("\three");
```

QUESTION 14

What is output by the code to the right?

A. `01011` B. `1011`

C. `10110` D. `00000`

E. `01000`

```
System.out.printf("%05d", 1011);
```

QUESTION 15

What is returned by the method call `something(4)`?

A. `4` B. `6`

C. `7` D. `8`

E. `9`

```
public static int something(int x) {
  x--;
  return x + x;
}
```

QUESTION 16

What is output by the code to the right?

A. `1` B. `3`

C. `4` D. `7`

E. There is no output due to a syntax error.

```
String pets = "Spot, Tiny Mo, Rex";
String[] chopped = pets.split("\\s");
System.out.print(chopped.length);
```

QUESTION 17

What replaces **<*1>** in the code to the right so that `main` is a valid main method for TestClass?

A. `main(Object[] args)`

B. `Object main(args[])`

C. `static void main(String[] args)`

D. `void main(Argument[] args)`

E. `static main(Object[] args)`

QUESTION 18

Assume **<*1>** is filled in correctly. What is output when the main method of `TestClass` is run?

A. `7`

B. `9`

C. `16`

D. There is no output due to a syntax error.

E. There is no output due to a runtime error.

```
public class TestClass {

  public <*1> {
    System.out.print(what(3, 4));
  }

  public static int what(int x) {
    return x * x;
  }

  public static int what(int x, int y) {
    return x + y;
  }

}
```

QUESTION 19

What is output by the code to the right?

A. `true true` B. `true false`

C. `false true` D. `false false`

E. There is no output due to a syntax error.

```
Scanner s = new Scanner("inputgoeshere");
System.out.print(s instanceof Scanner);
System.out.print(" ");
System.out.print(s instanceof Object);
```

QUESTION 20

How many combinations of values for the boolean variables `p`, `q`, `r`, and `s` will result in `t` being set to `true`?

A. `7` B. `12`

C. `16` D. `29`

E. `39`

```
boolean p, q, r, s;
// code to initialize p, q, r, and s
boolean t = (p && q) || (r && s)
```

QUESTION 21

What is output by the code to the right?

A. `8.0 9.0` B. `6.0 6.0`

C. `6 6` D. `3.0 2.0`

E. `2.3 3.2`

```
System.out.print(Math.pow(2,3));
System.out.print(" ");
System.out.print(Math.pow(3,2));
```

QUESTION 22

What is output by the code to the right?

A. `fun@ fun@` B. `Fun@ fun@`

C. `Fun! fun@` D. `fun! fun!`

E. `Fun! fun!`

```
String s1 = "Fun!";
String s2 = s1.toLowerCase();
System.out.print(s1 + " " + s2);
```

QUESTION 23

What is output by the code to the right?

A. `[4, 5, 1]`

B. `[1, 5, 4]`

C. `[4, 0, 5]`

D. `[5, 0, 4]`

E. `[4, 5, 0]`

```
ArrayList<Integer> list;
list = new ArrayList<Integer>();
list.add(4);
list.add(5);
list.add(1, 0);
System.out.print(list);
```

QUESTION 24

Which of the following is a valid Java variable name?

A. `x'` B. `x&y` C. `y!` D. `y111` E. More than one of these.

QUESTION 25

Which of these replaces **<*1>** in method `swap` to the right so the elements at indices `i` and `j` in array `arr` are swapped with each other?

A.
```
int temp = i;
i = j;
j = temp;
```

B.
```
int temp = arr[i];
arr[i] = arr[j];
arr[j] = temp;
```

C.
```
arr[j] = arr[j] ^ arr[i];
arr[i] = arr[j] ^ arr[i];
arr[j] = arr[j] ^ arr[i];
```

D.
```
arr[j] = arr[i] & arr[j];
arr[i] = arr[j] & arr[i];
```

E. More than one of these is correct.

```
public void sort(int[] arr) {
  boolean sorted = false;
  while(!sorted) {
    sorted = true;
    for(int j=0; j < arr.length-1; j++) {
      if (arr[j] > arr[j+1]) {
        swap(j, j+1, arr);
        sorted = false;
      }
    }
  }
}

public void swap(int i, int j, int[] arr) {
  <*1>
}
```

QUESTION 26

Assume **<*1>** is filled in correctly. Which type of sorting algorithm is implemented by `sort`?

A. Quick sort

B. Merge sort

C. Radix sort

D. Bubble sort

E. Dynamic sort

QUESTION 27

What is the largest possible value the code to the right will output?

A. `9.9`

B. `99`

C. `10`

D. `100`

E. `10000`

```
int total = 0;
double limit = Math.round(Math.random());
for(int i = 0; i < limit; i++) {
int temp = (int) (Math.random() * 100);
total += temp;
}
System.out.print(total);
```

QUESTION 28

What is output by the code to the right?

A. `ACB`

B. `BCA`

C. `ABC`

D. `B`

E. `A`

```
Queue<String> q = new LinkedList<String>();
q.add("A");
q.add("C");
q.add("B");
System.out.print(q.peek());
```

QUESTION 29

What replaces **<*1>** in the code to the right to indicate the `Map` has `String` keys and `Integer` values?

A. `<int><String>`

B. `<String><Int>`

C. `<Integer and String>`

D. `<String, Integer>`

E. `<<int>,<String>>`

```
Map<*1> map1 = new TreeMap<*1>();
map1.put("5", 1);
map1.put("3", 2);
map1.put("2", 3);

int sum = 0;
Iterator<Map.Entry<*1>> it;
it = map1.entrySet().iterator();
while(it.hasNext())
  sum += it.next().getValue();
System.out.print(sum);   // line 1

// test section
Map<*1> map2 = new TreeMap<*1>();
map2.put("2", 3);
map2.put("3", 2);
map2.put("5", 1);
if(map2.equals(map1))
  System.out.print("yes");
else
  System.out.print("no");
if(map2.hashCode() == map1.hashCode())
  System.out.print("yes");
else
  System.out.print("no");
```

For the remaining questions, assume that **<*1>** has been filled in correctly.

QUESTION 30

What is output by the line marked `// line 1`?

A. `532` B. `235` C. `1`

D. `3` E. `6`

QUESTION 31

What is output by the code after the `// test section` mark?

A. `yesyes` B. `yesno` C. `noyes`

D. `nono` E. The output will vary from one run of the program to the next.

QUESTION 32

On which of these arrays would static method `process()` return `35` if the array was passed as its parameter?

A. | 1 | 2 | 3 |

B. | 6 | 5 | 4 | 3 |

C. | 1 | 3 | 5 | 7 |

D. | 2 | 4 | 6 | 8 | 10 |

E. | 7 | 3 | 5 | 6 | 1 | 4 |

```
public static int process (int[] array) {
  int returnValue = 0;
  for (int i = 0; i < array.length; i++)
    for (int j = i; j < array.length; j++)
      returnValue += i;
  return returnValue;
}
```

QUESTION 33

What replaces **<*1>** and **<*2>** in the code to the right so that all of `s` up to but not including the space character is put in `firstName` and all of `s` after the space character is put in `lastName`? (Assume `s` has exactly one space character.)

A. **<*1>**: `s.substring(0, i)`
   <*2>: `s.substring(i, s.length())`

B. **<*1>**: `s.substring(0, i)`
   <*2>: `s.substring(i+1)`

C. **<*1>**: `s.substring(i-1)`
   <*2>: `s.substring(i+1, s.length())`

D. **<*1>**: `s.substring(i)`
   <*2>: `s.substring(i+1, s.length())`

E. **<*1>**: `s.substring(i, 0)`
   <*2>: `s.substring(s.length(), i+1)`

```
public class Student {

  public Student(String s) {
    firstName = getString(s, 1);
    lastName = getString(s, 2);
  }

  private static String
  getString(String s, int part) {
    int i;
    for(i=0; i<s.length(); i++) {
      if(s.charAt(i) == ' ')
        break;
    }
    return part==1 ? <*1> : <*2>;
  }

  <*3> {
    return firstName + " " + lastName;
  }

  private String firstName, lastName;
}

// client code
Student s = new Student("Stuart Little");
System.out.print(s);
```

QUESTION 34

Assume **<*1>** and **<*2>** are filled in correctly. What replaces **<*3>** in the code to the right so the client code outputs the string `Stuart Little`?

A. `public String toString()`

B. `public void outputString()`

C. `String String()`

D. `String getStringRepresentation()`

E. `protected String outputString()`

QUESTION 35

Which keyword is used in a method declaration to indicate the method may generate an exception that will not be handled within the method?

A. `throws` B. `may throw` C. `causes` D. `may cause` E. `may`

QUESTION 36

How many `*`'s are output by the code to the right?

A. `0` B. `4` C. `5`

D. `6` E. An infinite number since the loop never terminates.

```
int x = 5;
do {
  System.out.print('*');
} while (x-- > 0);
```

QUESTION 37

What replaces **<*1>** and **<*2>** in the code to the right so that class `Car`, class `Truck`, and the client code compiles without error?

A. **<*1>:** `public abstract`
   <*2>: `public void`

B. **<*1>:** `public`
   <*2>: `public abstract void`

C. **<*1>:** `protected`
   <*2>: `void`

D. **<*1>:** `private abstract`
   <*2>: `private void`

E. **<*1>:** `protected`
   <*2>: `public abstract void`

```
public abstract class Car {
  <*1> int mpg;
  <*2> setMpg(int newMpg);
}

public class Truck extends Car {
  public Truck() { }

  public void setMpg(int newMpg) {
    mpg = newMpg/2;
  }
}

// client code
Car v1 = new Truck();
v1.setMpg(30);
Truck v2 = new Truck();
v2.setMpg(30);
String s1 = v1.mpg+" "+v2.mpg;
System.out.print(s1);
```

QUESTION 38

Assume **<*1>** and **<*2>** are filled in correctly. What is output by the client code to the right?

A. `30 30` B. `30 15`

C. `15 15` D. `15 30`

E. There is no output due to a runtime error.

QUESTION 39

Which keyword is used in a class declaration to indicate that the class cannot be subclassed?

A. `uninheritable` B. `childless` C. `nosub` D. `const` E. `final`

QUESTION 40

Assume the method `mysort(int[] data)` is $O(N^2)$ where N is `data.length`. When `mysort` is passed an array of length 100,000 it takes 3 seconds to complete. If `mysort` is passed an array of length 500,000 what is the expected time in seconds it will take to complete?

A. `15` B. `25` C. `75` D. `2500` E. 25×10^{10}

2013-2014 Fall Computer Science Test 1
Answer Guide

1.	E	11.	D	21.	A	31.	A
2.	B	12.	E	22.	E	32.	E
3.	B	13.	C	23.	C	33.	B
4.	E	14.	A	24.	D	34.	A
5.	C	15.	B	25.	E	35.	A
6.	A	16.	C	26.	D	36.	D
7.	D	17.	C	27.	B	37.	B
8.	E	18.	A	28.	E	38.	C
9.	A	19.	A	29.	D	39.	E
10.	B	20.	A	30.	E	40.	C

2013-2014 Fall Computer Science Test 1
Answer Guide

1. **E.** One way to solve this problem is by adding one digit at a time. Add B_{16} plus D_{16} by converting B_{16} to 11 and D_{16} to 13, giving a sum of 24. In base 16, 24 is $1x16^1 + 8x16^0$, so 18_{16}. Leave the 8 in place as the last digit of the answer. Now, carry the 1 and add $1_{16} + A_{16} + C_{16}$, or 1 + 10 + 12, giving a sum of 23. In base 16, 23 is $1x16^1 + 7x16^0$, so 17_{16}. Leave the 7 in place as the second digit of the answer. Carry the 1, and the final answer becomes 178_{16}.
2. **B.** The order of operations in Java is the same as the order of operations in basic math – parentheses, exponents, multiplication, division, addition/subtraction. Hence, `m + 8 / 4 = 64 + (8 / 4) = 66`, and the value of `n` is 66. Then, `n / 3 + 6 = (66 / 3) + 6 = 28`, so the value of `p` is 28.
3. **B.** The code inside the `for` loop executes 10 times, when `i`=10, `i`=9, ... , `i`=1, and `i`=0. `total-=i` subtracts the current value of `i` from `total` on each iteration of the loop. `total` starts at 0, so its final value is 0 - 10 - 9 - 8 - 7 - 6 - 5 - 4 - 3 - 2 - 1 - 0 = -55.
4. **E.** This code will not compile due to a syntax error. The `charAt(int index)` method in the `String` class returns a `char`. Assigning a `char` to a `String` will cause a static type error.
5. **C.** When a Java array is created, the elements of the array are initialized to the "default" value for that type. For `boolean`, the default value is `false`, so `st[0]=false`, `st[1]=false`, and `st[2]=false`. Using the boolean OR operator `||` on `false`, `false`, and `false` yields `false`. The length of `st` is 3 as initialized in `new boolean[3]`.
6. **A.** The modulo operator `a%b` yields the remainder when `a` is divided by `b`. The statement `db%5%5` is performed from left to right, so first we find `5.5%5`, or the remainder when 5.5 is divided by 5, which is 0.5. Then applying the second modulo operator we find `0.5%5`, or the remainder when 0.5 is divided by 5, which is again 0.5.
7. **D.** Using DeMorgan's law, the expression `!(q >= p) || !(r < q)` can be transformed to `!((q >= p) && (r < q)`. `(q >= p)` is equivalent to `(p < q)`, so the expression can be simplified again to `!((p < q) && (r < q))` as given in the question.
8. **E.** A double multiplied or divided by an integer yields a double, so 1.11/2 = 0.555. The statement `if(one / 2 > 0)` hence evaluates to true. When an `if` statement is not followed by brackets containing statements to be executed, then the statement immediately after the `if` statement is executed, here `System.out.print("yes")`. Indentation of statements is not interpreted by Java. After `yes` is printed, execution moves to `System.out.print("no")` and `no` is printed. Finally the statement `if(one / 2 > 0.5)` evaluates to true, and a second `no` is printed for a total output `yesnono`.
9. **A.** The `String` class method `length()` returns the number of characters in the given string.
10. **B.** `dowhat` iterates through the given string, starting with the second character in the string at index 1, and counts the number of instances where the character being examined is "greater" than the character to its left. When a Java `char` is used as a number, such as in the `>` numerical comparison, that character's Unicode value is used for the computation. Hence `dowhat` counts the number of characters with a greater Unicode value than the character to the left. The characters `a` through `z` have the Unicode values 0x61 through 0x7A increasing, so `b > a` and `c > b`. The characters `0` through `9` have the Unicode values 0x30 through 0x39 increasing, so `2 > 1` and `3 > 2`. Since `1` is 0x31 and `c` is 0x63, it is not true that `1 > c`. Then the total number of characters in the string `abc123` which have a greater Unicode value than the character to the left is 4, and `dowhat("abc123")` returns 4.
11. **D.** The `Math` library method `Math.min(a, b)` returns the minimum of `a` and `b`. Here, -5 < 3, so -5 is the minimum value returned and is the value printed.
12. **E.** The Java expression `a & b` evaluates to the bitwise AND of `a` and `b`. Bitwise operators may only be applied to integer types `int`, `long`, `short`, `char`, and `byte`. `arr1` and `arr2` are `Object` references, not integer types, so attempting to apply `arr1 & arr2` will result in a static type error.
13. **C.** A character in a string beginning with a backslash `\` has special meaning and will alter the string output. The `\n` escape character inserts a newline at that place in the string. The `\t` character inserts a tab at that place in the string. Hence the first `print` statement results in two lines of output. The second `println` statement appends `two` to the second line and, since it is a `println` statement, adds a third line after printing the string. The third `print` statement appends a tab and `hree` to the third line. Hence there are three lines of output,

```
o
etwo
```

hree

14. **A.** The `printf` function takes a format string and one or more arguments to be used in the format string. The `%` character in the format string indicates the argument (here, `1011`) should be formatted according to the following part of the format string. The number before the decimal or format letter indicates the minimum "width" of the string to be outputted, and when prefaced by a 0 indicates the output will be zero-padded instead of padded with spaces, if necessary to meet the minimum width. Here, the output will be at least 5 characters and will be zero-padded in front. The `d` indicates the output will be formatted as a decimal integer. Hence, the output will be `1011` formatted as a 5-character zero-padded string, so `01011`.
15. **B.** The Java unary operator `a--` decrements the value of `a`. After the statement `x--`, `x` has the value 3, so returning `x + x` returns 6.
16. **C.** The `String` class method `split(String s)` splits the given string around matches of the regular expression `s`, putting each split section of the string into one element of the returned array. The regular expression matches are not included in the split sections of the string. The regular expression `\s` matches whitespace characters (another backslash before `\s` is necessary to prevent Java from interpreting `\` as the beginning of a string escape sequence, and instead to send the literal string `\s` to the regular expression interpreter). When `pets` is split around whitespace characters, the result is `chopped[0] = Spot,`, `chopped[1] = Tiny`, `chopped[2] = Mo,`, and `chopped[3] = Rex`. The length of `chopped` and the output will then be 4.
17. **C.** Every Java application must have a `main` method that is called when the application is started. The signature of the `main` method must be `public static void main(String[] args)`. Using `static public` instead of `public static` and using a different name for `args` is also permitted.
18. **A.** Java supports method overloading - defining more than one method of the same name in the same class - as long as the methods have different parameter lists (the number and type of parameters passed to the method). When an overloaded method is called in the code, the compiler infers which method is being called from the number and type of parameters passed to the method in the call. Here, the statement `what(3, 4)` is inferred to refer to the second `what` method, `what(int x, int y)` since it is the only `what` method that takes two integer parameters. This second `what` returns the sum of the two parameters, so the output value is 7.
19. **A.** The `a instanceof b` operator evaluates to true if object `a` is an instance of class `b`, an instance of a subclass of `b`, or an instance of a class that implements interface `b`. Here, `s` is an instance of class `Scanner`. Like all Java classes, `Scanner` is a subclass of `Object`, so `s` is also an instance of a subclass of `Object`, and both statements evaluate to true.
20. **A.** The value combinations of `p`, `q`, `r`, and `s` respectively for which both `p` and `q` are true and/or both `r` and `s` are true are: `true true true true`, `true true false true`, `true true true false`, `true true false false`, `true false true true`, `false true true true`, and `false false true true`. In total there are seven combinations of values which result in `t` set to `true`.
21. **A.** The `Math` library function `pow(a, b)` returns the double value a^b. Here, $2^3=8$ and $3^2=9$. Doubles are always printed with at least one digit to the right of the decimal, so the output is `8.0 9.0`.
22. **E.** The `String` function `toLowerCase()` returns the given string with upper-case alphabet characters converted to lower-case characters. Non-alphabetic characters are unchanged, so the value of `s2` is set to `fun!`. The method does not modify the original string, so at the time of the `print` statement, the value of `s1` remains `Fun!`. Together, the output is `Fun! fun!`.
23. **C.** The `ArrayList` funtion `add(e)` adds the element `e` to the end of the given `ArrayList`. So, after the first two calls to `add`, `list = [4, 5]`. The function `add(i, e)` adds the element `e` to the given `ArrayList` at index `i`, moving the elements at that index and higher one index to the right. Since `ArrayLists` are indexed beginning with 0, index 1 is the second element in list, and 0 will be inserted there, moving 5 one index to the right. Hence the final value of `list` is `[4, 0, 5]`.
24. **D.** Java variable identifiers must begin with an alphabetical (`a-z` or `A-Z`) character, underscore character `_`, or dollar sign character `$`. The first character may be followed by zero or more alphanumeric (`a-z`, `A-Z` or `0-9`), underscore or dollar sign characters. No other symbols can be used, so `x'`, `x&y`, and `y!` are all invalid variable names and will cause a syntax error during compilation. `y111meets` the requirements and is a valid variable name.
25. **E.** Answer choices B and C both correctly swap the two elements in the array. Choice C uses the XOR swap algorithm which uses a bitwise exclusive or operator `^` to swap the two elements. Using the associativity property of the XOR operator, we can deduce that after the second XOR, `arr[i] = (arr[j] ^ arr[i]) ^ arr[i] = arr[j] ^ (arr[i] ^ arr[i]) = arr[j] ^ 0 = arr[j]`. After the third XOR, `arr[j] = (arr[j] ^ arr[i]) ^ arr[j] = (arr[j] ^ arr[j]) ^ arr[i] = 0 ^ arr[i] = arr[i]`. (Note that in this explanation, `arr[i]` and `arr[j]` refer to the original values of `arr[i]` and `arr[j]` at the time swap was called.)

26. **D.** Bubble sort iterates through all the elements in the list, comparing adjacent elements and swapping them if they are out of order, causing larger values to "bubble" to the back of the list in an ascending sort like the one given here. The sort repeats this pass through the list until no adjacent elements are out of order, i.e. the list is sorted.
27. **B.** The `Math` library function `random()` returns a double in the range [0.0, 1). `Math.round(a)` rounds the double `a` to the nearest integer. The highest possible value for `limit` is 1, when `Math.random()` returns a value greater than or equal to 0.5 which is then rounded up to 1.0. The for loop will therefore execute at most once. `Math.random() * 100` will be at most 9.999… Casting a double to an int rounds down, so `temp` will be at most 99. Hence the final value of `total` will be at most 99.
28. **E.** The `add(e)` method of the `Queue` interface adds `e` to the beginning, or head, of the given queue, so after the three calls to `add`, `A` is at the head of the queue, `C` is second, and `B` is at the tail of the queue. `peek()` returns but does not remove the head of the given queue. Here, the call to `peek` will return `A`.
29. **D.** The `Map<K, V>` interface sets the generic type argument `K` as the type of the keys and `V` as the type of the values contained in the map. The correct way to declare a new map with `String` keys and `Integer` value is `Map<String, Integer>`.
30. **E.** The `entrySet()` method in the `Map` interface returns a `Set<Map.Entry<K,V>>`. The `iterator()` method in the `Set` interface returns an iterator over the `Map.Entry<K,V>` elements. The while loop iterates through each map entry, using the `getValue()` method in the `Map.Entry` interface to pull out the value in that key-value pair and add it to sum. The values in `map1` are the integers 1, 2, and 3; after the while loop is finished and these values have been added, `sum` will equal 6.
31. **A.** The `Map` interface overrides `equals` to be equivalent to `m1.entrySet().equals(m2.entrySet())`. `entrySet()` returns a `Set` of the key-value pairs, and the definition of `equals` for `Set` does not take order into consideration, so two maps will be equal if they have the same key-value pairs, regardless of the order in which the pairs were added. `map1` and `map2` contain the same key-value pairs, so `map2.equals(map1)` is true and yes will be printed. The `Map` interface also overrides `hashCode()` so that `m1.equals(m2)` implies `m1.hashCode() == m2.hashCode()`. Since `map1.equals(map2)`, then `map1.hashCode() == map2.hashCode()` and another `yes` is printed.
32. **E.** The `process` method never uses the values of `array`, only the length of the array. The array `{7,3,5,6,1,4}` has length 6, so the outer for loop iterates from 0 to 5 inclusive, and the inner for loop iterates from `i` to 5 inclusive, adding the value of `i` each time. On the first iteration of the outer for loop, `i=0`, so `returnValue` remains 0. On the second iteration, `i=1`, and the inner for loop iterates from 1 to 5, adding 1 each time, so `returnValue=5`. On the third iteration, `i=2`, and the inner for loop iterates from 2 to 5, adding 2 each time, so `returnValue=13`. On the fourth iteration, `i=3`, and the inner for loop iterates from 3 to 5, adding 3 each time, so `returnValue=22`. On the fifth iteration, `i=4`, and the inner for loop iterates from 4 to 5, adding 4 each time, so `returnValue=30`. On the sixth and final iteration, `i=5`, and the inner for loop executes once, adding 5 to `returnValue`, so the final value of `returnValue` is 35.
33. **B.** The construct `a ? b : c` evaluates to `b` if `a` is true, and evaluates to `c` if `a` is false, so **<*1>** should return `s` before the space character, corresponding to the call to `getString(s, 1)`, and **<*2>** should return `s` after the space character, corresponding to the call to `getString(s, 2)`. The `String` method `substring(begin, end)` returns the string beginning with the character at index `begin` and ending with the character at index `end-1`. After the for loop execution, `i` is the index of the space character in `s`, so to return all of `s` up to but not including the space character, `substring(0, i)` should be used. The `String` method `substring(begin)` returns the string beginning with the character at index `begin` through the end of the string. To return all of `s` after the space character, `substring(i+1)` should be used.
34. **A.** When an object is printed, its `toString()` method is called to determine the output to be printed. The default `Object` `toString` method prints the memory address of the object. For a class to print other output, the `toString` method must be overriden with a method with the header `public String toString()`.
35. **A.** The `throws` keyword indicates the method may cause an exception to be thrown which will not be handled within the method.
36. **D.** A postfix operator, such as `x--`, evaluates to the value of `x` before the decrement. The body of the do-while loop executes when `x=5, x=4, x=3, x=2, x=1`. When the body finishes and `x=1`, the statement `while(x-- > 0)` evaluates to `while(1 > 0)` because of the postfix operator rule, so the body is executed one more time. In total, the body executes 6 times, printing 6 stars.
37. **B.** Since `Truck` accesses `mpg`, it must be either `public` or `protected`. The client code also access `mpg`, so we can deduce it is a public variable. Variables cannot be declared `abstract`, so `mpg` cannot be `abstract`. Since

`setMpg()` does not have a method body defined in `Car`, it must be declared as an `abstract` method. Answer B is the only choice that satisfies these conditions.

38. **C.** When a subclass overrides a method implemented in its superclass, instances of the subclass always use the subclass's method definition. Since `v1` and `v2` are both instances of `Truck`, the `Truck` definition of `setMpg` will be the definition used, and both vehicles will have `mpg` set to 15.
39. **E.** The final keyword in Java indicates an entity cannot be modified or derived from. When used on a class, the class cannot be subclassed; on a method, the method cannot be overriden by inheriting classes; and on a variable, the value cannot be re-assigned.
40. **C.** In simplified terms, an algorithm which is $O(N^2)$ has a running time proportional to N^2 for large values of N, i.e. running time = αN^2. Thus the expected running time for an input 5 times as large as the original is $5^2=25$ times as long as the original running time.