UIL COMPUTER SCIENCE WRITTEN TEST

2025 STATE

MAY 2025

General Directions (Please read carefully!)

1. DO NOT OPEN THE EXAM UNTIL TOLD TO DO SO.
2. There are 40 questions on this contest exam. You will have 45 minutes to complete this contest.
3. All answers must be legibly written on the answer sheet provided. Indicate your answers in the appropriate blanks provided on the answer sheet. Clean erasures are necessary for accurate grading.
4. You may write on the test packet or any additional scratch paper provided by the contest director, but NOT on the answer sheet, which is reserved for answers only.
5. All questions have ONE and only ONE correct answer. There is a 2-point penalty for all incorrect answers.
6. Tests may not be turned in until 45 minutes have elapsed. If you finish the test before the end of the allotted time, remain at your seat and retain your test until told to do otherwise. You may use this time to check your answers.
7. If you are in the process of actually writing an answer when the signal to stop is given, you may finish writing that answer.
8. All provided code segments are intended to be syntactically correct, unless otherwise stated. You may also assume that any undefined variables are defined as used.
9. A reference to many commonly used Java classes is provided with the test, and you may use this reference sheet during the contest. AFTER THE CONTEST BEGINS, you may detach the reference sheet from the test booklet if you wish.
10. Assume that any necessary import statements for standard Java SE packages and classes (e.g., `java.util`, `System`, etc.) are included in any programs or code segments that refer to methods from these classes and packages.
11. NO CALCULATORS of any kind may be used during this contest.

Scoring

1. Correct answers will receive **6 points**.
2. Incorrect answers will lose **2 points**.
3. Unanswered questions will neither receive nor lose any points.
4. In the event of a tie, the student with the highest percentage of attempted questions correct shall win the tie.

STANDARD CLASSES AND INTERFACES – SUPPLEMENTAL REFERENCE

package java.lang

```
class Object
  boolean equals(Object anotherObject)
  String toString()
  int hashCode()

interface Comparable<T>
  int compareTo(T anotherObject)
    Returns a value < 0 if this is less than anotherObject.
    Returns a value = 0 if this is equal to anotherObject.
    Returns a value > 0 if this is greater than anotherObject.

class Integer implements Comparable<Integer>
  Integer(int value)
  int intValue()
  boolean equals(Object anotherObject)
  String toString()
  String toString(int i, int radix)
  int compareTo(Integer anotherInteger)
  static int parseInt(String s)

class Double implements Comparable<Double>
  Double(double value)
  double doubleValue()
  boolean equals(Object anotherObject)
  String toString()
  int compareTo(Double anotherDouble)
  static double parseDouble(String s)

class String implements Comparable<String>
  int compareTo(String anotherString)
  boolean equals(Object anotherObject)
  int length()
  String substring(int begin)
    Returns substring(begin, length()).
  String substring(int begin, int end)
    Returns the substring from index begin through index (end – 1).
  int indexOf(String str)
    Returns the index within this string of the first occurrence of str. Returns
    –1 if str is not found.
  int indexOf(String str, int fromIndex)
    Returns the index within this string of the first occurrence of str, starting
    the search at fromIndex. Returns –1 if str is not found.
  int indexOf(int ch)
  int indexOf(int ch, int fromIndex)
  char charAt(int index)
  String toLowerCase()
  String toUpperCase()
  String[] split(String regex)
  boolean matches(String regex)
  String replaceAll(String regex, String str)

class Character
  static boolean isDigit(char ch)
  static boolean isLetter(char ch)
  static boolean isLetterOrDigit(char ch)
  static boolean isLowerCase(char ch)
  static boolean isUpperCase(char ch)
  static char toUpperCase(char ch)
  static char toLowerCase(char ch)

class Math
  static int abs(int a)
  static double abs(double a)
  static double pow(double base, double exponent)
  static double sqrt(double a)
  static double ceil(double a)
  static double floor(double a)
  static double min(double a, double b)
  static double max(double a, double b)
  static int min(int a, int b)
  static int max(int a, int b)
  static long round(double a)
  static double random()
    Returns a double greater than or equal to 0.0 and less than 1.0.
```

package java.util

```
interface List<E>
class ArrayList<E> implements List<E>
  boolean add(E item)
  int size()
  Iterator<E> iterator()
  ListIterator<E> listIterator()
  E get(int index)
  E set(int index, E item)
  void add(int index, E item)
  E remove(int index)

class LinkedList<E> implements List<E>, Queue<E>
  void addFirst(E item)
  void addLast(E item)
  E getFirst()
  E getLast()
  E removeFirst()
  E removeLast()

class Stack<E>
  boolean isEmpty()
  E peek()
  E pop()
  E push(E item)

interface Queue<E>
class PriorityQueue<E>
  boolean add(E item)
  boolean isEmpty()
  E peek()
  E remove()

interface Set<E>
class HashSet<E> implements Set<E>
class TreeSet<E> implements Set<E>
  boolean add(E item)
  boolean contains(Object item)
  boolean remove(Object item)
  int size()
  Iterator<E> iterator()
  boolean addAll(Collection<? extends E> c)
  boolean removeAll(Collection<?> c)
  boolean retainAll(Collection<?> c)

interface Map<K,V>
class HashMap<K,V> implements Map<K,V>
class TreeMap<K,V> implements Map<K,V>
  Object put(K key, V value)
  V get(Object key)
  boolean containsKey(Object key)
  int size()
  Set<K> keySet()
  Set<Map.Entry<K, V>> entrySet()

interface Iterator<E>
  boolean hasNext()
  E next()
  void remove()

interface ListIterator<E> extends Iterator<E>
  void add(E item)
  void set(E item)

class Scanner
  Scanner(InputStream source)
  Scanner(String str)
  boolean hasNext()
  boolean hasNextInt()
  boolean hasNextDouble()
  String next()
  int nextInt()
  double nextDouble()
  String nextLine()
  Scanner useDelimiter(String regex)
```

STANDARD CLASSES AND INTERFACES – SUPPLEMENTAL REFERENCE

Package java.util.function

```
Interface BiConsumer<T,U>
  void accept(T t, U u)

Interface BiFunction<T,U,R>
  R apply(T t, U u)

Interface BiPredicate<T,U>
  boolean test(T t, U u)

Interface Consumer<T>
  void accept(T t)

Interface Function<T,R>
  R apply(T t)

Interface Predicate<T>
  boolean test(T t)

Interface Supplier<T>
  T get()
```

UIL COMPUTER SCIENCE WRITTEN TEST – 2025 STATE

Note: Correct responses are based on **Java SE Development Kit 22 (JDK 22)** from Oracle, Inc. All provided code segments are intended to be syntactically correct, unless otherwise stated (e.g., "error" is an answer choice) and any necessary Java SE 22 Standard Packages have been imported. Ignore any typographical errors and assume any undefined variables are defined as used. **For all output statements, assume that the System class has been statically imported using: `import static java.lang.System.*;`**

Question 1

Which of the following is equivalent to the expression $2857_9 + 132_5$?

A) 2210_{10} **B)** 4230_8 **C)** 2814_9 **D)** 33300_5 **E)** None are equivalent.

Question 2

What is output by the code to the right?

A) 16 **B)** 5 **C)** 38 **D)** 92

E) There is no output due to a compile error.

```
out.print(243+34 / 21-9 * 21-17);
```

Question 3

What is output by the code to the right?

A) 1.2YEPZ **B)** 1.2YepY

C) 1.3YEPY **D)** 1.3YepZ

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

```
String s = "Yep";
int i = 89;
double d = 1.25;
out.printf("%3$.1f%1$S%2$c",s,i,d);
```

Question 4

What is output by the code to the right?

A) −17 **B)** −1 **C)** 17 **D)** 1

E) There is no output due to a compile error.

```
String s = "CaptHanSolo";
String r = "CaptHandsome";
int i = s.compareTo(r);
out.println(i);
```

Question 5

What is output by the code to the right?

A) true

B) false

```
boolean a = false;
boolean b = true;
a |= a ^ b & !b | a;
b ^= b ^ !a & a | !b;
out.print(a ^ b | a);
```

Question 6

What is output by the code to the right?

A) 4.00 4.00 **B)** 4.50 4.49

C) 4.50 4.50 **D)** −3.50 −3.5

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

```
double c = 4.5;
c = Math.nextAfter(c, -7);
String s = "" + c;
s = s.substring(0, 4);
out.printf("%.2f %s",c,s);
```

Question 7

What is output by the code to the right?

A) 12−7

B) 13−7

C) 2−7

D) 13−6

E) 3−6

```
int i = -7;
if(i < 0 | i++ > -10)
     out.print(1);
if(i > 0 && i-- < 10)
     out.print(2);
else out.print(3);
out.print(i);
```

Question 8

What is output by the code to the right?

A) 54 **B)** 63 **C)** 31 **D)** 22

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

```
int i = 212;
i &= i - 7 << 4;
i %= i ^ 123 - 17;
out.print(i | 13);
```

Question 9

What is output by the line marked `//q09` in the code to the right?

A) `[13, 13, 15, 19, 25]`

B) `[8, 8, 10, 14, 20]`

C) `[17, 15, 15, 17, 21]`

D) `[12, 10, 10, 12, 16]`

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

Question 10

What is output by the line marked `//q10` in the code to the right?

A) `[8, 6, 9, 3, 20]`

B) `[8, 9, 8, 6, 20]`

C) `[12, 8, 9, 5, 16]`

D) `[8, 8, 9, 5, 20]`

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

```
int[][] mat = new int[3][5];
for(int r = 0; r < 3; r++)
    for(int c = 4; c >= 0; c--)
    mat[r][c] =
         (r + 1) * 5 + (c - 1) * c - 2;
out.println
    (Arrays.toString(mat[2])); //q09
for(int i = 0; i < 4; i ++) {
    if(i != 3) {
        mat[i][i] = mat[i][i + 1];
        mat[i][i]--;
    } if(i != 0) {
        mat[i - 1][i + 1] =
            mat[i - 1][i - 1];
        mat[i - 1][i]++;
    }
    mat[0][i] += i;
    mat[1][i] -= i;
}
out.println
    (Arrays.toString(mat[1])); //q10
```

Question 11

Which of the following could replace **<1*>** for the code to the right to compile without error?

A) `IOException`

B) `FileNotFoundException`

C) `Exception`

D) Only A or C.

E) Any of the above will work.

```
public void scan()throws <1*>{
    File f = new File("a.dat");
    Scanner sc = new Scanner(f);
    sc.nextLine();
    out.print(sc.nextLine());
}
```

Question 12

What is output by the code to the right?

A) `4`

B) `104`

C) `15`

D) `105`

E) There is no output due to a compile error.

```
int i = 4;
switch(5) {
    case 4: i += 10;
    case 2: i++;
        break;
    default: i += 100;
}
out.print(i);
```

Question 13

What is the order of precedence for the operators to the right?

A) `III, II, I, IV`

B) `I, III, IV, II`

C) `III, I, II, IV`

D) `III, I, II, IV`

E) `III, I, IV, II`

```
I. ==
II. +=
III. <=
IV. ?: (ternary)
```

Question 14

What is output by the code to the right?

A) `5`

B) `3`

C) `8`

D) `6`

E) There is no output due to a compile error.

```
int[] sizes = new int[] {
    Double.BYTES, Float.BYTES,
    Long.BYTES, Integer.BYTES,
    Short.BYTES, Byte.BYTES
};
Arrays.sort(sizes);
out.print(sizes[0]+sizes[2]);
```

Question 15

What is output by the code to the right?

A) `[Ironhide, RC, Bumblebee, RC, Optimus]`

B) `[Ironhide, RC, RC, Optimus]`

C) `[Ironhide, RC, Optimus]`

D) `[Ironhide, Bumblebee, RC, Optimus]`

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

```
ArrayList<String> a;
a = new ArrayList<String>();
a.add("Bumblebee");
a.add("Optimus");
a.add(1,"RC");
a.set(0, "Ironhide");
if(!a.contains(new String("RC")))
     a.add(1, "RC");
out.println(a);
```

Question 16

What is output by the code to the right?

A) `true i`

B) `false 0`

C) `false NaN`

D) `true NaN`

E) There is no output due to a compile error.

```
double x = 0, y = 0;
x /= y--;
y = Math.sqrt(y);
out.print((y == x) + " ");
out.println(y);
```

Question 17

What is output by the following client code?

```
out.print(A.B.s);
```

A) `C`

B) `D`

C) Output cannot be determined until runtime.

D) There is no output due to a compile error.

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

```
class A {
     static class B {
          static String s = "C";
     }
     static TopLevel B = new TopLevel();
}
class TopLevel { String s = "D"; }
```

Question 18

What is the asymptotic time complexity of the code to the right?

A) $\mathcal{O}(n \log n)$

B) $\mathcal{O}(\log n)$

C) $\mathcal{O}(n)$

D) $\mathcal{O}(n^2)$

E) $\mathcal{O}(n^2 \log n)$

Question 19

Which data structure is demonstrated by the code to the right?

A) Queue

B) Min-Heap

C) Linked List

D) Max-Heap

E) Stack

```
PriorityQueue<Integer> pq;
pq = new PriorityQueue<Integer>
     (Collections.reverseOrder());
for(int y = 0; y < N; y++) {
     double d = Math.random() * 100;
     pq.add((int)(d));
}
out.println(pq);
```

Question 20

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 runtime error.

```
String m1 = "(\\w+ ){2,7}";
String m2 = "([A-Z]?[a-z]* ?)+";
String s = "One of a Kind";
out.print(s.matches(m1));
out.print(" ");
out.print(s.matches(m2));
```

Question 21

Which of the following could replace **<1*>** in the client code to the right so that the resulting code produces a compile-time error?

A) `<T>`

B) `<? extends T>`

C) `<R>`

D) Nothing (leave the space blank)

E) A or C

Question 22

Suppose we were to implement the solution proposed in option A in question 21 in the code to the right. In addition, we replace all instances of `Node` with `Node<T>` (except for the occurrences of `Node` on the marked lines). Which of the following Java concepts is demonstrated by the resulting code?

A) Hiding

B) Overloading

C) Overriding

D) Encapsulation

E) Abstraction

Question 23

Suppose we were to implement the solution proposed in option C in question 21 in the code to the right. In addition, we replace all instances of `Node` with `Node<R>` (except for the occurrences of `Node` on the marked lines). Would this new code produce any warnings or errors?

A) The code would demonstrate the same concept as in Question 22 and not result in an error.

B) The code would not demonstrate the same concept as in Question 22 but would result in an error.

C) The code would demonstrate the same concept as in Question 22 and result in an error.

D) The code would not demonstrate the same concept as in Question 22 and not result in an error, but would not work as intended.

E) The code would not demonstrate the same concept as in Question 22 and not result in an error, and would work as intended.

Question 24

The class `DataStructure` in the code to the right is a partial implementation of what well-known data structure?

A) Dequeue

B) AVL Tree

C) Stack

D) Min-Heap

E) Queue

```
class DataStructure<T> {
    private class Node<1*> { // EXCEPT THIS LINE
        public T data;
        public Node next, prev;

        public Node(T data) { // EXCEPT THIS LINE
            this.data = data;
        }
    }

    private Node head, tail;
    private int size;

    public DataStructure() {
        head = tail = null;
        size = 0;
    }

    public void addFront(T data) {
        Node newNode = new Node(data);
        if (head == null) {
            tail = newNode;
        } else {
            newNode.next = head;
            head.prev = newNode;
        }
        head = newNode;
        size++;
    }

    public void addBack(T data) {
        Node newNode = new Node(data);
        if (tail == null) {
            head = newNode;
        } else {
            newNode.prev = tail;
            tail.next = newNode;
        }
        tail = newNode;
        size++;
    }

    public T removeFront() {
        if (head == null) {
            return null;
        }
        T data = head.data;
        head = head.next;
        if (head == null) {
            tail = null;
        } else {
            head.prev = null;
        }
        size--;
        return data;
    }

    public T removeBack() {
        if (tail == null) {
            return null;
        }
        T data = tail.data;
        tail = tail.prev;
        if (tail == null) {
            head = null;
        } else {
            tail.next = null;
        }
        size--;
        return data;
    }
}
```

Question 25

Using the undirected graph to the right, what would be the length of its minimum spanning tree?

A) 27 **B)** 33 **C)** 35 **D)** 43 **E)** 50

Question 26

Which of the following algorithms can be used to calculate the length of the minimum spanning tree for the graph to the right?

A) Kruskal's Algorithm **B)** Floyd–Warshall Algorithm

C) Ford–Fulkerson Algorithm **D)** Prim's Algorithm

E) Both A and B. **F)** Both A and C.

G) Both A and D.

Question 27

Among the different variants of the algorithm, which of the following is equivalent to the best-known asymptotic time complexity of Kruskal's Algorithm?

A) $\mathcal{O}(V \log V)$ **B)** $\mathcal{O}(E \log V)$

C) $\mathcal{O}(E)$ **D)** $\mathcal{O}(V^2)$

E) None of the above.

Question 28

What is the shortest cost path in the graph to the right between nodes A and E?

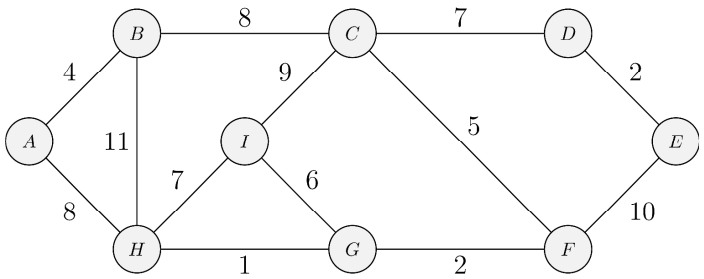
A) 17

B) 21

C) 25

D) 27

E) 33



Question 29

Which of the following boolean expressions, when evaluated over all permutations of `true` and `false`, is equivalent to the truth table to the right?

A) `!C || !B && (!A || C)`

B) `(!A && !B && !C) || A`

C) `(!A && C) || (A && B)`

D) `(A && !C) || (!A && !B)`

E) More than one of the above.

A	B	C	Output
F	F	F	T
F	F	T	T
F	T	F	F
F	T	T	F
T	F	F	T
T	F	T	F
T	T	F	T
T	T	T	F

Question 30

What is the value of the expression to the right?

A) 1024 **B)** 2048

C) 2047 **D)** 8192

E) None of the above.

$2^{23} = 8388608$

Find the value of $8388608 \gg 10$

/* Use the code below to answer question 31 through 36 */

```
final class Character {
    final int hp;
    final String name;
    final double basicAttackDamage;
    final double secondaryAttackDamage;
    final double primaryAbilityDamage;
    final double secondaryAbilityHeal;
    final double ultimateAbilityDamage;
    final int basicAttackCoolDownMillis;
    final int secondaryAttackCoolDownMillis;
    final int primaryAbilityCoolDownMillis;
    final int secondaryAbilityCoolDownMillis;
    final int ultimateAbilityCoolDownMillis;
    final double ultimateChargeRate;
    final double ultimateChargePerDamage;

    public Character(CharacterThing thing) {
        this.hp                              = thing.hp;
        this.name                            = thing.name;
        this.basicAttackDamage               = thing.basicAttackDamage;
        this.secondaryAttackDamage           = thing.secondaryAttackDamage;
        this.primaryAbilityDamage            = thing.primaryAbilityDamage;
        this.secondaryAbilityHeal            = thing.secondaryAbilityHeal;
        this.ultimateAbilityDamage           = thing.ultimateAbilityDamage;
        this.basicAttackCoolDownMillis       = thing.basicAttackCoolDownMillis;
        this.secondaryAttackCoolDownMillis   = thing.secondaryAttackCoolDownMillis;
        this.primaryAbilityCoolDownMillis    = thing.primaryAbilityCoolDownMillis;
        this.secondaryAbilityCoolDownMillis = thing.secondaryAbilityCoolDownMillis;
        this.ultimateAbilityCoolDownMillis   = thing.ultimateAbilityCoolDownMillis;
        this.ultimateChargeRate              = thing.ultimateChargeRate;
        this.ultimateChargePerDamage         = thing.ultimateChargePerDamage;
    }

    @Override
    public String toString() {
        Field[] fields = this.getClass().getDeclaredFields();
        StringBuilder res = new StringBuilder();

        for (Field field : fields) {
            try {
                res.append((field.get(this)) != null ? field.getName() + " = " +
                                                       field.get(this).toString() + "\n" : "");
            } catch (Exception ignored) {}
        }

        return res.toString().trim();
    }

    static class CharacterThing {
        private int hp;
        private String name;
        private double basicAttackDamage;
        private double secondaryAttackDamage;
        private double primaryAbilityDamage;
        private double secondaryAbilityHeal;
        private double ultimateAbilityDamage;
        private int basicAttackCoolDownMillis;
        private int secondaryAttackCoolDownMillis;
        private int primaryAbilityCoolDownMillis;
        private int secondaryAbilityCoolDownMillis;
        private int ultimateAbilityCoolDownMillis;
        private double ultimateChargeRate;
        private double ultimateChargePerDamage;
```

/* Continued on Next Page */

```
static {
  out.println("You sure can.");
}
{
 this.name = "Good Luck";
 out.println("Wait... you can do this?");
}

public static CharacterThing thingy() {
    return new CharacterThing();
}

<2*> CharacterThing() {}

public CharacterThing hp(int hp) {
    this.hp = hp;
    return this;
}

public CharacterThing name(String name) {
    this.name = name;
    return this;
}

public CharacterThing basicAttackDamage(double basicAttackDamage) {
    this.basicAttackDamage = basicAttackDamage;
    return this;
}

public CharacterThing secondaryAttackDamage(double secondaryAttackDamage) {
    this.secondaryAttackDamage = secondaryAttackDamage;
    return this;
}

public CharacterThing primaryAbilityDamage(double primaryAbilityDamage) {
    this.primaryAbilityDamage = primaryAbilityDamage;
    return this;
}

public CharacterThing secondaryAbilityHeal(double secondaryAbilityHeal) {
    this.secondaryAbilityHeal = secondaryAbilityHeal;
    return this;
}

public CharacterThing ultimateAbilityDamage(double ultimateAbilityDamage) {
    this.ultimateAbilityDamage = ultimateAbilityDamage;
    return this;
}

public CharacterThing basicAttackCoolDownMillis(int basicAttackCoolDownMillis) {
    this.basicAttackCoolDownMillis = basicAttackCoolDownMillis;
    return this;
}

public CharacterThing secondaryAttackCoolDownMillis(int secondaryAttackCoolDownMillis) {
    this.secondaryAttackCoolDownMillis = secondaryAttackCoolDownMillis;
    return this;
}

public CharacterThing primaryAbilityCoolDownMillis(int primaryAbilityCoolDownMillis) {
    this.primaryAbilityCoolDownMillis = primaryAbilityCoolDownMillis;
    return this;
}
```

/* Continued on Next Page */

```
        public CharacterThing secondaryAbilityCoolDownMillis(int secondaryAbilityCoolDownMillis) {
            this.secondaryAbilityCoolDownMillis = secondaryAbilityCoolDownMillis;
            return this;
        }

        public CharacterThing ultimateAbilityCoolDownMillis(int ultimateAbilityCoolDownMillis) {
            this.ultimateAbilityCoolDownMillis = ultimateAbilityCoolDownMillis;
            return this;
        }

        public CharacterThing ultimateChargeRate(double ultimateChargeRate) {
            this.ultimateChargeRate = ultimateChargeRate;
            return this;
        }

        public CharacterThing ultimateChargePerDamage(double ultimateChargePerDamage) {
            this.ultimateChargePerDamage = ultimateChargePerDamage;
            return this;
        }

        public Character thing() {
            return new Character(<1*>);
        }
    }
}

////////////////////////////////////// client code ////////////////////////////////////////
Character character_1 = Character.CharacterThing.thingy()
        .hp(76_67_77)
        .name("Terry")
        .basicAttackDamage(212)
        .secondaryAttackDamage(131)
        .primaryAbilityDamage(1413.14)
        .secondaryAbilityHeal(1512.74)
        .basicAttackCoolDownMillis(300)
        .primaryAbilityCoolDownMillis(2500)
        .secondaryAttackCoolDownMillis(1000)
        .secondaryAbilityCoolDownMillis(4750)
        .ultimateAbilityDamage(9999)
        .ultimateAbilityCoolDownMillis(9000)
        .ultimateChargeRate(0.015)
        .ultimateChargePerDamage(0.045)
        .thing();

Character character_2 = new Character.CharacterThing()
        .hp(10_000)
        .ultimateAbilityDamage(7231)
        .basicAttackDamage(149)
        .secondaryAttackDamage(342)
        .thing();

out.println(character_1); // line 1
out.println(character_2); // line 2
```

Question 31

The `CharacterThing` class in the source code above demonstrates which of the following design patterns?

A) Singleton

B) Object Pool

C) Builder

D) Prototype

E) Factory Method

Question 32

Which of the following concepts is utilized in the client code to the right?

A) Overriding

B) Overloading

C) Abstraction

D) Method Chaining

E) Encapsulation

```
Character character_2 = new
    Character.CharacterThing()
        .hp(212_212)
        .name("Stacey")
        .ultimateAbilityDamage(7231)
        .basicAttackDamage(212)
        .secondaryAttackDamage(342)
        .thing();
```

Question 33

Which of the following could replace **<1*>** in the code above to ensure objects are created with the provided values?

A) `null`

B) `CharacterThing.thingy()`

C) `this`

D) Nothing is required.

E) More than one of the above.

Question 34

Which of the following could replace **<2*>** in the code above to ensure all client code runs successfully in the following question?

A) `protected`

B) `public`

C) `void`

D) Nothing is required.

E) More than one of the above.

Question 35

Assuming any errors in the client code above have been corrected, what is output to the console when the code marked `//line 1` has finished executing?

A)
```
You sure can.
Wait... you can do this?
hp = 766777
name = Terry
basicAttackDamage = 212.0
secondaryAttackDamage = 131.0
primaryAbilityDamage = 1413.14
secondaryAbilityHeal = 1512.74
ultimateAbilityDamage = 9999.0
basicAttackCoolDownMillis = 300
secondaryAttackCoolDownMillis = 1000
primaryAbilityCoolDownMillis = 2500
secondaryAbilityCoolDownMillis = 4750
ultimateAbilityCoolDownMillis = 9000
ultimateChargeRate = 0.015
ultimateChargePerDamage = 0.045
```

B)
```
Wait... you can do this?
You sure can.
hp = 766777
name = Good Luck
basicAttackDamage = 212.0
secondaryAttackDamage = 131.0
primaryAbilityDamage = 1413.14
secondaryAbilityHeal = 1512.74
ultimateAbilityDamage = 9999.0
basicAttackCoolDownMillis = 300
secondaryAttackCoolDownMillis = 1000
primaryAbilityCoolDownMillis = 2500
secondaryAbilityCoolDownMillis = 4750
ultimateAbilityCoolDownMillis = 9000
ultimateChargeRate = 0.015
ultimateChargePerDamage = 0.045
```

C)
```
Wait... you can do this?
Wait... you can do this?
You sure can.
hp = 766777
name = Good Luck
basicAttackDamage = 212.0
secondaryAttackDamage = 131.0
primaryAbilityDamage = 1413.14
secondaryAbilityHeal = 1512.74
ultimateAbilityDamage = 9999.0
basicAttackCoolDownMillis = 300
secondaryAttackCoolDownMillis = 1000
primaryAbilityCoolDownMillis = 2500
secondaryAbilityCoolDownMillis = 4750
ultimateAbilityCoolDownMillis = 9000
ultimateChargeRate = 0.015
ultimateChargePerDamage = 0.045
```

D)
```
You sure can.
Wait... you can do this?
Wait... you can do this?
hp = 766777
name = Terry
basicAttackDamage = 212.0
secondaryAttackDamage = 131.0
primaryAbilityDamage = 1413.14
secondaryAbilityHeal = 1512.74
ultimateAbilityDamage = 9999.0
basicAttackCoolDownMillis = 300
secondaryAttackCoolDownMillis = 1000
primaryAbilityCoolDownMillis = 2500
secondaryAbilityCoolDownMillis = 4750
ultimateAbilityCoolDownMillis = 9000
ultimateChargeRate = 0.015
ultimateChargePerDamage = 0.045
```

E) There is no output due to an error

Question 36

Assuming any errors in the client code above have been corrected, what is output to the console when the code marked `//line 2` has finished executing, ignoring all output by previous lines?

A)
```
hp = 10000
name = Good Luck
basicAttackDamage = 149.0
secondaryAttackDamage = 342.0
ultimateAbilityDamage = 7231.0
```

B)
```
You sure can.
Wait... you can do this?
hp = 10000
name = Good Luck
basicAttackDamage = 149.0
secondaryAttackDamage = 342.0
ultimateAbilityDamage = 7231.0
```

C)
```
Wait... you can do this?
You sure can.
hp = 10000
name = Good Luck
basicAttackDamage = 149.0
secondaryAttackDamage = 342.0
primaryAbilityDamage = 1413.14
secondaryAbilityHeal = 1512.74
ultimateAbilityDamage = 7231.0
basicAttackCoolDownMillis = 300
secondaryAttackCoolDownMillis = 1000
primaryAbilityCoolDownMillis = 2500
secondaryAbilityCoolDownMillis = 4750
ultimateAbilityCoolDownMillis = 9000
ultimateChargeRate = 0.015
ultimateChargePerDamage = 0.045
```

D)
```
You sure can.
Wait... you can do this?
hp = 10000
name = null
basicAttackDamage = 149.0
secondaryAttackDamage = 342.0
primaryAbilityDamage = 0.0
secondaryAbilityHeal = 0.0
ultimateAbilityDamage = 7231.0
basicAttackCoolDownMillis = 0
secondaryAttackCoolDownMillis = 0
primaryAbilityCoolDownMillis = 0
secondaryAbilityCoolDownMillis = 0
ultimateAbilityCoolDownMillis = 0
ultimateChargeRate = 0.0
ultimateChargePerDamage = 0.0
```

E)
```
hp = 10000
name = Good Luck
basicAttackDamage = 149.0
secondaryAttackDamage = 342.0
primaryAbilityDamage = 0.0
secondaryAbilityHeal = 0.0
ultimateAbilityDamage = 7231.0
basicAttackCoolDownMillis = 0
secondaryAttackCoolDownMillis = 0
primaryAbilityCoolDownMillis = 0
secondaryAbilityCoolDownMillis = 0
ultimateAbilityCoolDownMillis = 0
ultimateChargeRate = 0.0
ultimateChargePerDamage = 0.0
```

F)
```
hp = 10000
name = null
basicAttackDamage = 149.0
secondaryAttackDamage = 342.0
primaryAbilityDamage = 0.0
secondaryAbilityHeal = 0.0
ultimateAbilityDamage = 7231.0
basicAttackCoolDownMillis = 0
secondaryAttackCoolDownMillis = 0
primaryAbilityCoolDownMillis = 0
secondaryAbilityCoolDownMillis = 0
ultimateAbilityCoolDownMillis = 0
ultimateChargeRate = 0.0
ultimateChargePerDamage = 0.0
```

Question 37

Which of the following statements is not true about protected methods?

A) They can be accessed from within the class it was defined in.

B) They can be accessed from a different class within same package as the protected method.

C) They can be accessed from any subclass of the class it's contained in, within the same package as the protected method.

D) They can be accessed from any subclass of the class it's contained in, from a different package than the protected method.

E) None of the above.

Question 38

Which of the following pairings does not properly match the data structure in question to the worst-case asymptotic time complexity of the operation on that structure?

A) $\mathcal{O}(1)$ random access using an Array.

B) $\mathcal{O}(1)$ find min using a Min-Heap

C) $\mathcal{O}(\log n)$ search using a Tree Set

D) $\mathcal{O}(1)$ search using a Hash Set.

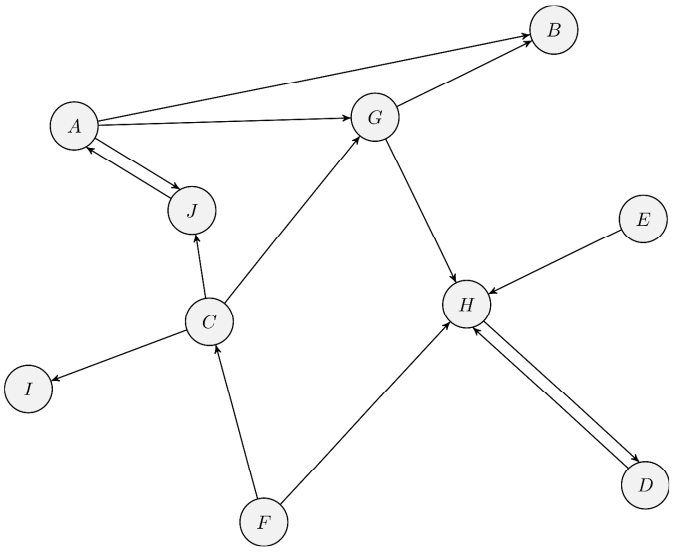
E) $\mathcal{O}(\log n)$ search using a Binary Search Tree

Question 39

Consider a series of four (potentially identical) binary search trees created from 20 arbitrary, but distinct (no duplicate) values. Let the first tree be created such that the width of the tree is maximized. We will denote the width of that tree as $w_{\max}$. Let the second tree be created such that the width of the tree is minimized. We will denote the width of that tree as $w_{\min}$. Let the third tree be created such that the height of the tree is maximized. We will denote the height of that tree as $h_{\max}$. Let the fourth tree be created such that the height of the tree is minimized. We will denote the height of that tree as $h_{\min}$. Your answer should be these four values separated by commas in the order of $w_{\max}$, $w_{\min}$, $h_{\max}$, $h_{\min}$. When calculating the height or width of the tree, assume that the number of edges are being counted, and not the number of vertices.

Question 40

Consider the adjacency matrix representation of the graph to the right. How many times does the value "0" appear in the adjacency matrix?



★ANSWER KEY – CONFIDENTIAL★

UIL COMPUTER SCIENCE – 2024-2025 STATE

Questions (+6 points for each correct answer, -2 points for each incorrect answer)

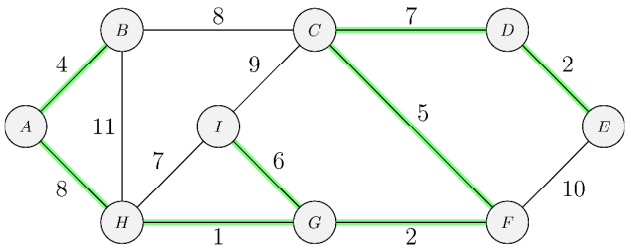
1) B	11) E	21) B	31) C
2) C	12) B	22) A	32) D
3) C	13) E	23) B	33) C
4) A	14) A	24) A	34) E
5) B	15) C	25) C	35) D
6) B	16) C	26) G	36) E
7) D	17) B	27) B	37) E
8) C	18) A	28) B	38) E
9) A	19) D	29) D	*39) 9,1,19,4
10) D	20) C	30) D	*40) 86

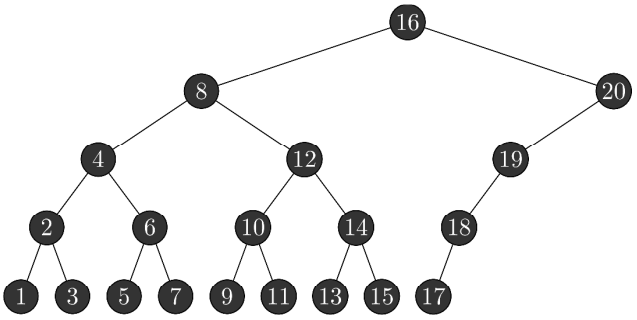
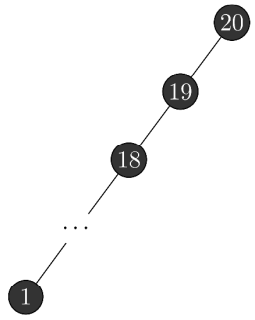
** See "Explanation" section below for alternate, acceptable answers.*

Note: Correct responses are based on **Java SE Development Kit 22 (JDK 22)** from Sun Microsystems, Inc. All provided code segments are intended to be syntactically correct, unless otherwise stated (e.g., "error" is an answer choice) and any necessary Java SE 22 Standard Packages have been imported. Ignore any typographical errors and assume any undefined variables are defined as used.

Explanations:

1.	B	Basic base conversion problem.
2.	C	Basic order of operations problem.
3.	C	`%c` will output a character with the ASCII value given. The number before the `$` in a `printf` term denotes which of the following objects will be output by the term, it is 1-indexed.
4.	A	`String`'s `compareTo` method will return the ASCII difference between the first two characters that are different between the two `String`s.
5.	B	Simple boolean solving.
6.	B	`Math.nextAfter(a, b)` will return the closest number to `a` in the direction of `b` (either positive or negative), so `4.49999999999`… in this case.
7.	D	Double boolean operators (`||`, `&&`), will short-circuit (if the first value is `false` for an `&&`, the second will not be evaluated because it is impossible for the expression to be `true`. Similar for the first value of a `||` expression being `true`). Single boolean operators do not so this.
8.	C	Order of operations with Java expressions.
9.	A	Matrix Tracing.
10.	D	Matrix Tracing.
11.	E	Only a `FileNotFoundException` is required, but `FileNotFoundException` extends `IOException` which extends `Exception`, so any of the 3 will work.
12.	B	Literals are allowed in switches, but since none of the cases are true only the default statement will execute.
13.	E	Java Order of Precedence.
14.	A	After sorting: `[1, 2, 4, 4, 8, 8]`, bytes is simply size divided by 8, and then it will be sorted. Then 1 and 4 will be chosen from positions 0 and 2.
15.	C	Simple `ArrayList` tracing, an `ArrayList` of `String`s `contains()` method will use `.equals()` to compare, not == so the actual content will be compared, not the memory addresses.
16.	C	`NaN` is never equal to `NaN`, and both values will be `NaN` once the operations are complete.
17.	B	Instance variables take precedence over inner classes with the same name.
18.	A	The runtime of adding to a max-heap is $\mathcal{O}(\log n)$, and it is being done n times.
19.	D	Java's `PriorityQueue` utilizes a min-heap ordinarily, but because it is initialized with the `Collections.reverseOrder()` object, it is actually implementing a max-heap in this case.
20.	C	First `String m1` will match 2-7 words followed by a space, the `String s` does not end with a space. Second `String m2` will match one or more instances of a capital letter followed by 0 or more lowercase letter followed by 0 or 1 spaces, which matches.
21.	B	The notation of `<? extends T>` is a common pattern to be used when creating an instance of data structure that all extend a certain class `T`, but not within the definition of a class.
22.	A	Doing this will create a compile-time warning. The exact warning is as follows: "The type parameter `T` is hiding the type `T`" While also a compile-timer error produced by many IDEs, Java officially identifies this issue known as Type Hiding, which occurs when a class is re-defined within a more specific scope, thus hiding the existence of the class defined in the earlier scope.
23.	B	Doing this will create multiple compile-time errors. The first of these reveals the real issue. The exact wording of the first error is as follows: "`R` cannot be resolved to a type" Note that `R` is defined within the localized scope of the `Node` class. Thus, the `DataStructure` class does not know of its existence since it was not defined within that scope or a higher scope. Note that this is unique from the issue of type hiding, as the type of `R` is simply not accessible to the outside class due to issues with scope.
24.	A	The `DataStructure` class is an implementation of a `Dequeue` which can be described as a combination of a `Stack` and a `Queue`. Namely, an $\mathcal{O}(1)$ insertion and removal operation from the front of the structure (`Queue`-like) and the back of the structure (`Stack`-like).

25.	C	Consider the following cover of edges that is a minimum spanning tree:  There are multiple minimum spanning trees for this graph, but this one has a length of 35.
26.	G	Both Prim's and Kruskal's algorithms are algorithms for calculating the minimum spanning tree of a graph.
27.	B	Since $\|E\| = \mathcal{O}(V^2)$, we can note that $\mathcal{O}(E \log E) = \mathcal{O}(E \log V)$. This is the time complexity for Kruskal's algorithm.
28.	B	The shortest path is either $AHGFE$ or $ABCDE$, both of which are of cost 21.
29.	D	Simple boolean table/expression evaluation/comparison
30.	D	Note that $8388608 \gg 10 \equiv 2^{23} \gg 10 \equiv 2^{23-10} \equiv 2^{13} \equiv 8192$
31.	C	This is a classic example of the Builder design pattern
32.	D	This is a classic example of Method Chaining
33.	C	The `this` keyword is the only option that keeps functionality. Option A compiles but leads to a runtime exception, and option B compiles and executes but results in an empty object being created every time with only default values.
34.	E	Because of the client code marked `//line2`, the constructor must be reachable from the client code which is assumed to be in the same package. This means that answer choices A, B, and C all work.
35.	D	Because both objects are created before being printed, the `static` block and setup block both execute before the code marked `//line1` is executed. Then the object prints with the values set when creating the object.
36.	E	Since both objects have already been created and printed as part of the previous line, the output here is only the object. The object prints all declared variables, and unassigned variables print the primitive default values. name is set as part of the initializer block that was executed when the object was created.
37.	E	Protected methods, instance variables, constructors, and classes can be accessed by entities with the following scopes: anything (i) within the same class (ii) within a subclass contained in the same package (iii) within a class contained in the same package (iv) within a subclass contained within a different package. The only situation that an entity cannot access a protected member is if it is from a different class (not subclass) in a different package. The four options A through D are equivalent to (i) through (iv), thus, the answer is none of the above.
38.	E	While it is true that search in a binary search tree is $\Omega(\log n)$ (i.e., best case), it is not true that it is $\mathcal{O}(\log n)$ (i.e., worst case). Binary search trees have an $\mathcal{O}(n)$ search operation since, in the worst case, a binary search tree can effectively be a linked list.

39.	9,1,19,4	Note that using the values 1 through 20 is a valid set of 20 arbitrary and distinct values. Any other set of 20 values can be mapped to the values 1 through 20 based on their lexicographical order. Therefore, we will assume that the values are 1 through 20 for the sake of creating concrete examples of the four trees. Consider the following set of trees:  $w_{\mathbf{max}} = 9$ & $h_{\mathbf{min}} = 4$  $w_{\mathbf{min}} = 1$ & $h_{\mathbf{max}} = 19$ In general, h_{min} occurs when the tree is complete. That said, in this case, the proposed tree also works. For w_{max}, we can note that a complete tree here only has a width of 8 while the tree proposed has a width of 9. Note that attempting to increase this width would require a 21^{th} node. For the w_{min} and h_{max} cases, these both happen when the tree is effectively a linked list.
40.	86	An adjacency matrix for $\|V\|$ vertices requires $\|V\|^2$ space to represent the edges E. Since we have $\|V\| = 10$, we know that the adjacency matrix will take up $10^2 = 100$ numbers (either a 1 for an edge being present, or a 0 for an edge not being present). There are a total of 14 edges present, so 14 of the 100 values will be a 1. This leaves $100 - 14 = 86$ values that will be 0.