# Unit 4 – ICS4U0 – Objects & ADTs

Sample Test – Wednesday November 5, 2025

Names: Gorski

| Total | Knowledge | Communication | Thinking | Application |
|---|---|---|---|---|
| | | | | |
| (100) | (23) | (22) | (25) | (30) |

## Knowledge

1. Identify the object method type in each **bolded** method call by writing in the appropriate first letter. /8
(C)onstructor, (M)utator, (A)ccessor, (F)acilitator

| | |
|---|---|
| F | (a) int num = (int) Math.**random**()*5; ← not one of category |
| F | (b) FileOutputStream out = **openFileOutput**("txt.txt", Activity.MODE_PRIVATE); |
| C | (c) EditText et = **new EditText**(); |
| A | (d) String words = et.**getText**(); |
| M | (e) textArea.**setText**("Hello"); |
| F | (f) if(question.**equals**(answer)) |
| F | (g) if(word.**compareTo**(secondWord)>0) |
| A | (h) while(!s**isFull**()) is = accessor |

2. What 11 methods signatures would be needed by the Food class? Fill in the blanks below. /11

A Food object has 3 instance variables: String name (e.g. cookie), int calories (e.g. 1234), double price (e.g. 4.99).

Constructors

```
public Food ()
public Food (String n, int c, double p)
```

Accessors

```
public String getName()
public int getCalories()
public double getPrice()
public String toString()
```

Mutators

```
public void setName (String n)
public void setCalories (int c)
public void setPrice (double p)
```

Others

```
public boolean equals (Food f)
public int compareTo (Food f)
```

3. What does each acronym stand for? /4

OOP: Object Oriented Programming

ADT: Abstract Data Type

LIFO: Last In First Out

FIFO: First In First Out

AEIH: Abstraction Encapsulation Information Hiding

# Communication

4. In the first column, fill in the term that matches the description. /12

| | |
|---|---|
| instance | (a) Variables found inside an object's class. |
| private | (b) A keyword restricting variable access to within the class. |
| class | (c) A template for an object or type. Also contains a java program. |
| accessor | (d) An object method type that returns values of instance variables. |
| constructor | (e) An object method type that sets up dynamic memory. |
| mutator | (f) An object method type that changes the values of instance variables. |
| Encapsulation | (g) Keeping an object's code self-contained and independent of other code. It relies only on itself. |
| object | (h) A general term for data and the methods associated with that data. |
| O(1) | (i) The Big Oh notation time of peek. |
| O(1) | (j) The Big Oh notation time of dequeue. |
| Abstraction | (k) Other programmers can use your objects without deep understanding of the specifics of how it is coded. |
| Stack (or Queue) | (l) An example of an ADT we studied in class. |

5. Choose the best data structure for each description. /5

Stacks
- Recursive calls
- Undo Button

Queues
- Printer Jobs
- Banking Transactions
- Boarding a Plane

| | | |
|---|---|---|
| Stack (Queue) | (a) | A FIFO structure. |
| (Stack) Queue | (b) | A LIFO structure. |
| (Stack) Queue | (c) | A pile of books models this data structure well. |
| Stack (Queue) | (d) | The data structure that could model a waiting line. |
| (Stack) Queue | (e) | The data structure used by the browser's back button. |

6. Why is an constructor an odd method compared to others inside an object? (2 sentences) /2

① When you call a constructor, you need to add the word "new" in front of it. No other methods do this.
② A constructor has no return type. It returns one of itself.
③ A constructor must have the same name as the class.

7. What three characteristics of a class support information hiding? (3 sentences, explain each briefly) /3

① All instance variables are private to hide their information
② All changes must go through mutators so changes are strictly controlled, keeping code stable.
③ All access to variables must go through accessors so content is regulated.

# Thinking

8. Neighbours share a long, narrow, driveway. Cars parked in the driveway leave by backing out. A schedule makes sure that nobody is blocked in when they need to leave. On each day, departing cars leave before any cars enter. Before Monday, no cars are in the driveway. The table shows how the driveway is shared. /2

The driveway at the end of Monday is shown below:

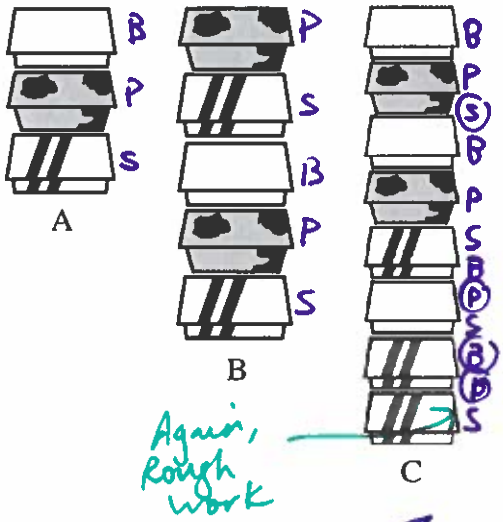

V R F D R B K B A

I always trace them out with diagram

| Day | # Leaving | # Entering | Owners in enter order |
|---|---|---|---|
| Mon | 0 | 2 | Ariadne, Bob |
| Tues | 1 | 3 | Kate, Ben, Roy |
| Wed | 2 | 1 | Daisy |
| Thurs | 0 | 2 | Finn, Rose |
| Fri | 3 | 1 | Vincent |

Whose cars will be parked at the end of Friday? (circle)
(A) Bob, Vincent, Daisy
(B) Vincent, Ariadne, Rose
(C) Ariadne, Kate, Vincent
(D) Ariadne, Daisy, Vincent

9. Alan and Turing are working at a burger restaurant. Alan cooks burgers one at a time. After cooking a burger, he places it into one of three different boxes: one with stripes, one with a pattern and one plain box. If he has cooked three burgers, he would have a stack as shown in diagram A. If he cooked two more burgers, he would have a stack as shown in diagram B. /2




Again, Rough work

As Alan cooks a burger, he places that box on the top of the stack of not yet sold burgers and continues to cycle through the three different boxes (stripe, pattern, plain, stripe, pattern, plain, ...) to place the burger. Turing sells the burgers one at a time and always takes the uppermost box from the stack. Alan is cooking faster than Turing can sell the burgers. After some time, Turing has sold some burgers and Alan has cooked more burgers.

Suppose the stack of unsold burgers looks as shown in diagram C. What is the fewest number of burgers sold by Turing? (circle)
(a) 4
(b) 5
(c) 6
(d) 7

10. Note what is printed out beside each line of output. /6

more tracing

```
Stack s = new Stack ();          4 3 2 1
s.push (1);
s.push (2);
s.push (3);
System.out.println (s.pop ());   3
s.push (4);
System.out.println (s.size ());  3
s.push (5);
System.out.println (s.pop ());   5
System.out.println (s.peek ());  4
System.out.println (s.pop ());   4
```

more tracing.

```
Queue q = new Queue ();          5 4 3 2 1
q.enqueue (1);                   F
q.enqueue (2);
q.enqueue (3);
System.out.println (q.dequeue ());  1
q.enqueue (4);
System.out.println (q.size ());     3
q.enqueue (5);
System.out.println (q.dequeue ());  2
System.out.println (q.peek ());     3
System.out.println (q.dequeue ());  3
```

11. Consider the following Fruit class:

```
public class Fruit {

   private int c;

   public Fruit() {
      c = 15;
   }
   public Fruit(int m) {
      c = m;
   }
   public int mash() {
      return (c - 5);
   }
```

doesn't change

```
public int getCount() {
      return c;
}
public void setCount(int m){
      c = m;
}
public void decrease() {
      c = c - 2;
}
public boolean equals(Fruit f) {
     if(c == f.getCount ())
       return true;
     else
       return false;
}
```

does change

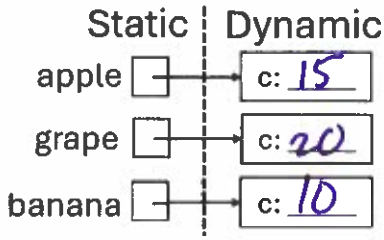
```
public String toString() {
   return "Count: "+ c;
}

public int compareTo(Fruit f){
   if (c > f.getCount())
      return 1
   else if (c < f.getCount())
      return -1
   else
      return 0;
}}
```

oops.

(a) Trace this class: fill in both the output (on the lines). You may use the diagram to track the values. /6

```
Fruit apple = new Fruit ();
System.out.println(apple.toString ());
System.out.println(apple.mash ());
System.out.println(apple.toString());
Fruit grape = new Fruit (20);
System.out.println(grape);
System.out.println(grape.equals (apple));
Fruit banana = new Fruit (10);
System.out.println(banana.compareTo (grape));
System.out.println(grape.compareTo (banana));
```

Count: 15
10
count: 15
Count: 20
F
-1
1




(b) Write the code to call the Fruit class. /4

| | |
|---|---|
| Make a new Fruit, an orange. It has a count of 902. | Fruit orange = new Fruit (902); |
| Call the decrease method on your orange. | orange.decrease(); ← don't forget brackets |
| Make a new Fruit, a plum. It calls the default constructor. | Fruit plum = new Fruit (); |
| Print out the plum's values on the screen. | System.out.println(plum); |

12. This code would run the above Fruit class. Circle and correct 5 errors. /5

```
Fruit peach = new Peach ();

Fruit cherry = new Fruit (80);

System.out.println(peach);

cherry.setCount();

System.out.println(peach.getCount() + " peaches");

if (peach.equals(cherry))
    System.out.println ("They are the same");
```

Fruit

new

# for example 3

don't forget to cross out the semi colon

# Application

13. (a) Fill in the Name class (instance variables: first and last name), /20

```
// Class line
public class Name {
  // Instance Variables
  private String first;
  private String last;
//❶&❷Constructors
public Name () {
  first = "Eve";
  last = "Ning";
}
public Name (String f, String L){
  first = f;
  last = L;
}
//❸Mutator
public void setFirst (String f)
{
  first = f;
}
public void setLast (String L)
{
  last = L;
}
//❹Accessor
public String getFirst(_____ __)
{
  return first;
}
public String getLast(_____ __)
{
  return last;
}
//❺To String (Accessor)
public String toString(_____ __)
{
  return "Name[value=" + first
  Can vary    + " " + last+ "]";
}
//❻CompareTo, use last name
public int compareTo(Name n)
{
 if(last.compareTo(n.getLast())>0)
    return 1;
 else if(last.compareTo(n.getLast())<0)
    return -1;
 else
    return 0;
}
```

* question says last name

```
//Returns their initials
public String getInitals(_____ __){
  return first.charAt(0)
       + last.charAt(0);
}
//❼Equals (Facilitator) use first name
public boolean equals (Name n)
{
if(first.equals(n.getFirst()))
    return true;
  else
    return false;
}
}
```

* question says first name

(b) Adapt the Stack Class to make a Stack of Names.

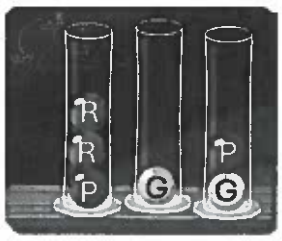
```
public class NameStack {
  private int count;
  private Name data[] = new Name [50];
    public NameStack () {
        count = 0; }
    public void push (Name addMe) {
        data [count] = addMe;
        count++; }
    public int size () {
        return count; }
    public boolean isFull () {
        return (count == 50); }
    public Name pop () {{
        count--;
        return data [count]; }
    public Name peek () {
        return data [count--]; }
    public boolean isEmpty () {
        return count == 0; }
    public void clear () {
        count = 0; }
}
```

Remember:
- 5 objects in stack
- 6 objects in Queue

(c) Create a new Stack of names, add two names to it.

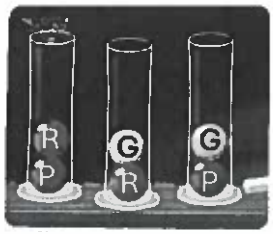
```
NameStack s = new NameStack();
Name n = new Name("Sara", "Bellum");
s.push(n);
Name m = new Name("Ida", "Knowe");
s.push(m);
```

14. (a) Fill in the code to make these 3 Stacks.



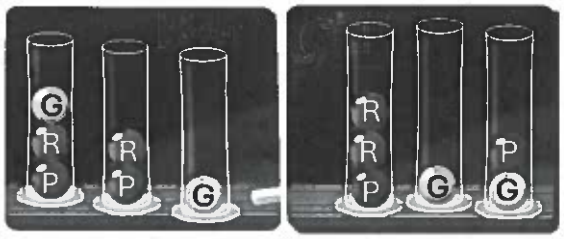
```
s1.push("P");
s1.push("R");
s1.push("R");

s2.push("G");

s3.push("G");
s3.push("P");
```

(b) Fill in the code. Change the previous to this picture.



```
s3.push(s2.pop());
s2.push(s1.pop());
s2.push(s3.pop());

s1.push(s3.pop());
s2.push(s3.pop());
s3.push(s1.pop());
s3.push(s2.pop());
```

(c) What is the smallest number of steps you can take to transfer the stack in the first picture to the second?



Do not use a loop.
Only push and pop.

/4

Smallest number of steps?.. 5 .....

15. Flip the bottom 2 elements of the Stack named 's'. After you are done, the rest of the Stack should be in its original order. The picture shows **an example** of this process.

Do not redeclare and push elements onto the Stack, remember the picture is **an example**. Anything could be in the Stack named 's'. In fact, nothing could be in the stack.

/6

```
//if the stack named s is less than 2 elements, write error
if( s.size() < 2 )
    System.out.println("Error");
else {
    //make stack s2
    Stack s2 = new Stack();
    //reverse s's elements into s2
    while ( !s.isEmpty() ) {        <- the common Stack loop.
        s2.push(s.pop());
    }
    //switch the top two elements of s2
    //pop an element, store it in a variable
    int temp = s2.pop();
    //do the other things needed to switch the top two element of s2
        s.push(s2.pop());
        s.push(temp);

    //reverse s2's elements into s
    while ( !s2.isEmpty() ) {
        s.push(s2.pop());
    }
}
```