Unit 4 – ICS4U0 – Objects & ADTs

Sample Test – Tuesday April 8, 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;
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**()) don't forget 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 cost (e.g. 4.99).

Constructors

public Food ()

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

Accessors

public String getName ()

public int getCalories ()

public double getCost ()

public String toString ()

Mutators

public void setName (String n)

public void setCalories (int c)

public void setCost (double c)

Others

public boolean equals (Food f)

public int compareTo (Food f)

3. What does each acronym stand for? /4

Object	Abstract	Last	First
Oriented	Data	In	In
Programming	Type	First	First
		Out	Out

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

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. What is the trade-off associated with a Stack? /2

⊕ All Stack operations (push, pop, peek, isEmpty) are O(1). This is very very fast.

⊖ A Stack is limited to LIFO operations

Trade off: With a Stack, we compromise on functionality to gain speed.

7. What are three characteristics of a class that supports information hiding? Explain each briefly. /3

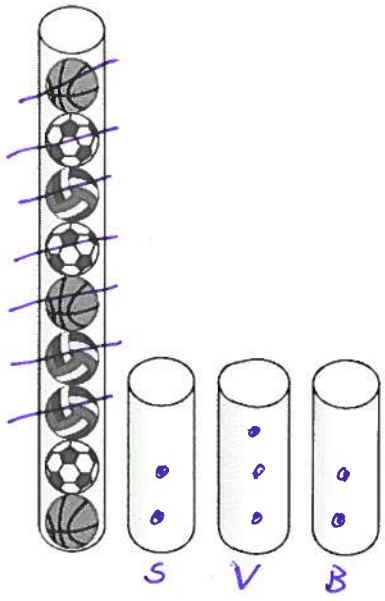
① All instance variables are private to hide their information

② All change 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. An equipment tube contains basketballs, volleyballs, soccer balls. Zara reorganizes the equipment by moving the balls one by one. That is, Zara takes each ball out of the top of the equipment tube and then drops it into the top of one of the smaller tubes. After moving each ball once, each type of ball is stored together in a smaller tube. /2



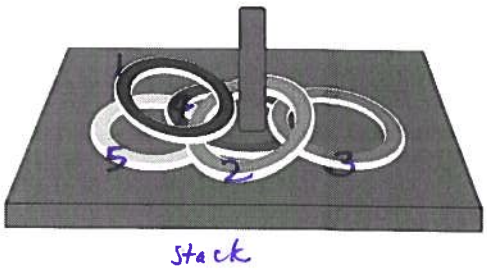
All the balls are the same size and each smaller tube holds three balls. Which of the following statements is true? (circle the correct answer).

a) Zara fills the tube of soccer balls first.
b) Zara fills the tube of volleyballs last.
c) Zara fills the tube of soccer balls after filling the tube of basketballs.
(d) Zara fills the tube of volleyballs before filling the tube of basketballs.

9. Nia tries to throw five rings around a peg as part of a game. The following chart shows how points are earned each time a ring lands around the peg. Rings that do not land around the peg do not earn points. /2

The following picture illustrates the order in which Nia threw her five rings. It also shows which ones landed around the peg and which ones did not land around the peg.

Toss	Points
First Toss	5 miss
Second Toss	(4) ✓
Third Toss	3 miss
Fourth Toss	(2) ✓
Fifth Toss	1 miss



How many points did Nia earn? (circle)

a) 15 b) 9 (c) 6 d) 3

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

```
Stack s = new Stack ();          8 4 3 2 1 5
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
```

```
Queue q = new Queue ();
q.enqueue (1);                   1 2 3 4 5
q.enqueue (2);                   Front  Q  Back
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
```

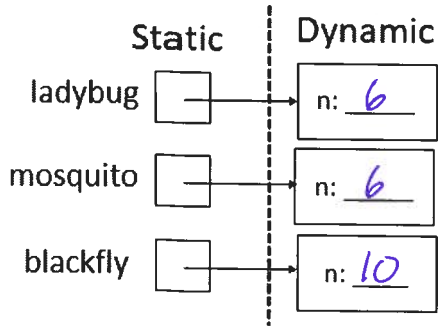
11. Consider the following bug class:

```
public class bug {

   private int n;

   public bug () {
      n = 6;
   }
   public bug (int r) {
      n = r;
   }
   public int buzz () {
      return (n + 2);
   }

   public int fly () {
      return n;
   }
   public void setFly(int r){
      n = r;
   }
   public void minusTwo () {
      n = n-2;
   }
   public boolean equals (bug b)
   {
     if(n == b.fly ())
       return true;
     else
       return false;
   }

   public String toString () {
     return "bug="+ n;
   }

   public int compareTo (bug b)
   {
      if (n == b.fly ())
         return 0;
      else if (n > b.fly ())
         return 1;
      else
         return -1;
   }}
```

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

```
bug ladybug = new bug ();
System.out.println(ladybug.toString ());          bug=6
System.out.println(ladybug.fly ());               6
bug mosquito = new bug (6);
System.out.println(mosquito);                     bug=6      (automatically calls toString)
System.out.println(mosquito.equals (ladybug));    true
bug blackfly = new bug (10);
System.out.println(blackfly.compareTo (mosquito)); 1
System.out.println(mosquito.compareTo (blackfly)); -1
```

(me: 6, them: 10)



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

Make a new bug, a cricket. It has a value of 82.	bug cricket = new bug (82);
Call the minusTwo method on your cricket.	cricket.minusTwo();
Make a new bug, an aphid. It calls the default constructor.	bug aphid = new bug ();
Print out the aphid's values on the screen.	System.out.println(aphid); (or aphid.toString() or aphid.fly())

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

```
bug earwig = new bug ();

bug boxelder = new bug (0);

System.out.println(earwig.fly()); ---- or eliminate System.out.println

boxelder.minusTwo();

if (boxelder.compareTo (earwig) > 0)
     System.out.println (boxelder + " is bigger");
else
     System.out.println (earwig + " is bigger");
```

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
                + " " + last + "]";
}
//❷CompareTo, use last name
public int compareTo (Nam n)
{
 if(last.compareTo(n.getLast())>0)
    return 1;
 else if(last.compareTo(n.getLast())<0)
    return -1;
 else
    return 0;
}
```

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

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

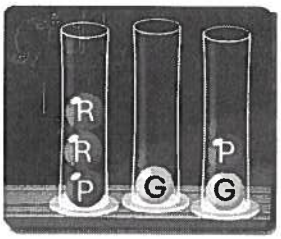
```
public class NameStack {
  private int count;
  private Object(Name) data[] = new Object(Name) [50];
    public NameStack () {
        count = 0; }
    public void push (Object(Name) addMe)  {
        data [count] = addMe;
        count++; }
    public int size ()  {
        return count; }
    public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

Remember
-50 objects in Stack
-6 objects in Queue
need changing

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

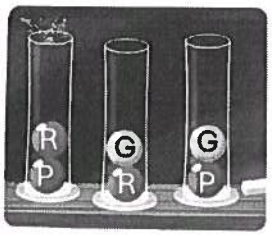
```
NameStack s = new NameStack();
Name n = new Name("Sara","Bellum");
s.push(n);
Name m = new Name("Ida","Knowe");
s.push(m);
```

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



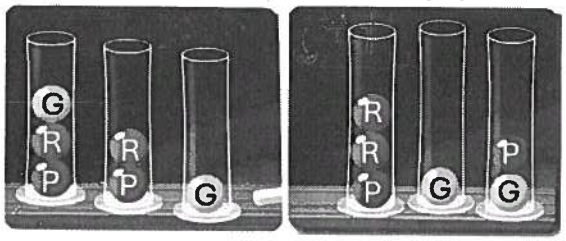
```
s1.push("P");
s1.push("R");
s1.push("R");

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()) {
        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());
    }
}
```