Unit 4 – ICS4U0 – Objects & ADTs

Sample Test – Thursday November 7, 2024

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 tree class? Fill in the blanks below. /11

A Tree has 3 instance variables: int height, char type (c=coniferous, d=deciduous), String name (e.g. maple).

Constructors
public Tree ()
public Tree (int h, char t, String n)

Accessors
public int getHeight()
public char getType()
public String getName()
public String toString()

Mutators
public void setHeight(int h)
public void setType(char t)
public void setName(String n)

Others
public int compareTo (Tree t)
public boolean equals (tree t)

3. What does each acronym stand for? /4

Object	Abstract	Last	First
Oriented	Data	In	In
Programming	Type	First	First
		Out	Out
		(stack)	(Queue)

Communication

4. In the first column, fill in the terms that match 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.
itself or nothing	(l) The constructor's return type.

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 model 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). That is constant time, which is very, very fast.

⊖ A Stack is limited to LIFO operations

$\boxed{\text{Trade Off}}$ 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

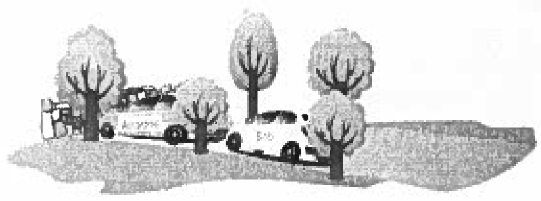
② All changes must go through mutators, so changes are strictly controlled, keeping code stable.

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:



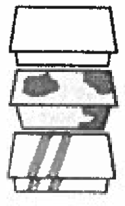
*Draw to trace it.

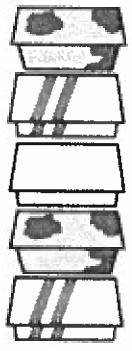
V
~~K~~
~~D~~
~~K~~
~~B~~
K
~~B~~
A
Drive

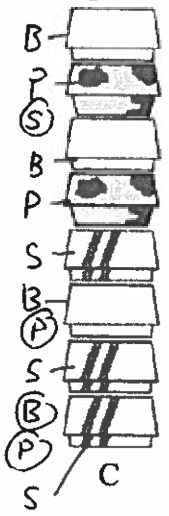
Day	# Leaving	# Entering	Owners in enter order
Mon	0	2	Ariadne, Bob
Tues	1	3	Kate, Ben, Roy
Wed	2	1	Daisy
Thurs	0	2	Finn, Rose
Fri	3	1	Vincent

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


A


B



C

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.

S / P / B

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

*Draw them to trace it

$\cancel{9}$ $\cancel{8}$ $\cancel{7}$ 6
Front Back

$\cancel{6}$
$\cancel{7}$
$\cancel{8}$
9
Bottom

```
Queue q = new Queue ();
q.enqueue (9);
q.enqueue (8);
System.out.println (q.dequeue ());      9
System.out.println (q.size ());         1
q.enqueue (7);
q.enqueue (6);
System.out.println (q.dequeue ());      8
System.out.println (q.dequeue ());      7
System.out.println (q.isEmpty ());      false
```

```
Stack s = new Stack ();
s.push (9);
s.push (8);
System.out.println (s.pop ());          8
System.out.println (s.size ());         1
s.push (7);
s.push (6);
System.out.println (s.pop ());          6
System.out.println (s.pop ());          7
System.out.println (s.isEmpty ());      false
```

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);
   }
```

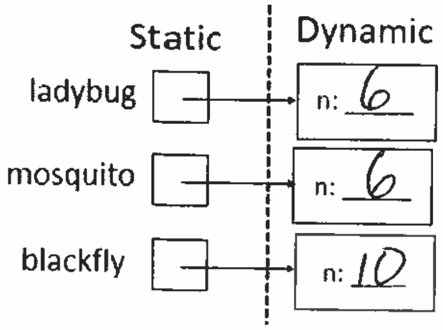
```
public int fly () {
   return n;
}
public void setFly(int r){
   n = r;
}
public void minusTwo () {
   n = n-2;
}
public boolean equals (bug b) {
  if(n == b.fly ())
    return true;
  else
    return false;
}
```

```
public String toString () {
  return "bug:[fly="+ 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:[fly=6]
System.out.println(ladybug.fly ());                   ...6....
bug mosquito = new bug (6);
System.out.println(mosquito);                         bug:[fly=6]
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....
```



(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())

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

```
                       bug
bug earwig = new earwig ();

bug boxelder = new bug (2);
                                     fly
System.out.println(earwig.setB(12));

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 ;
//①②Constructors
public Name () {
  first = "Eve" ;
  last = "Ning" ;
}
public Name (String f, String L) {
  first = f ;
  last = L ;
}
//①Mutator
public void setFirst (String f)
{
  first = f ;
}
public void setLast (String L)
{
  last = L ;
}
//①Accessor
public String getFirst ( )
{
 return first ;
}
public String getLast ( )
{
 return last ;
}
//①To String (Accessor)
public String toString ( )
{
 return "Name[Value=" + first
          + " " + last+"]";
}
//②CompareTo, 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 ;
}
```

```
//Returns their initials .
public String getInitials ( ) {
  return first.charAt(0)
         + last.charAt(0) ;
}
//①Equals (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.

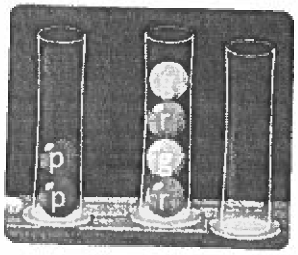
```
public class NameStack {
  private int count;
  private ~~Object~~ Name data[] = new ~~Object~~ Name [50];
    public NameStack () {
       count = 0; }
    public void push (~~Object~~ Name addMe)   {
       data [count] = addMe;
       count++; }
    public int size ()  {
       return count; }
    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; }
}
```

- 50 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);
```

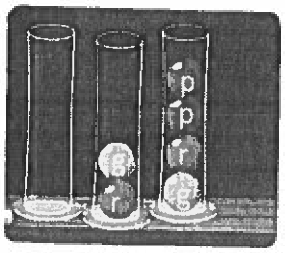
14. (a) Fill in the code to make these 3 Stacks.



```
s1.push("_p_");
s1.push("_p_");

s2.push("_r_");
s2.push("_g_");
s2.push("_r_");
s2.push("_g_");
```

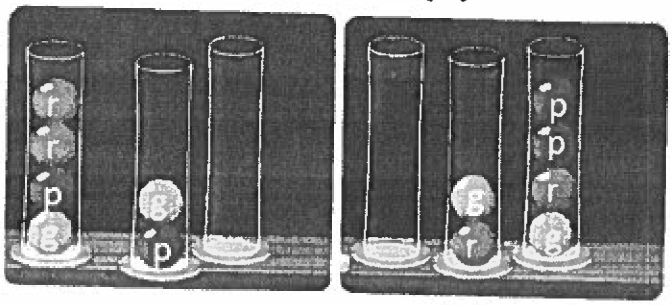
(b) Fill in the code. Change the previous to this picture.



```
s3.push(_s2_.pop());
s3.push(_s2_.pop());

s3.push(_s1_.pop());
s3.push(_s1_.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.

/5

Smallest number of steps?...6......

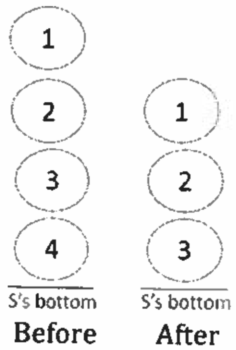
15. Remove and print out the bottom element of the Stack named 's'. After you are done, the rest of the Stack should be in its original order. The picture shows **an example** of this process.

you need loops because you don't know what's there

Notes:
(1) **Use a loop.**
(2) Do not redeclare and push elements onto the Stack, remember the picture is **an example**.
(3) Anything could be in the Stack named 's'. In fact, nothing could be in the stack.

/5

```
if(! s.isEmpty())          <- to handle nothing
{
    Stack s2 = new Stack();         -- to make a new stack to hold it temperarily
    while(! s.isEmpty())
        s2.push(s.pop());           ] — the standard stack loop ☆
    System.out.println(s2.pop());
    while(! s2.isEmpty())
        s.push(s2.pop());           ]
}
```



Before After

Prints: 4