THIS EXAMINATION WILL NOT BE DEPOSITED IN THE LIBRARY RESERVE

BROCK UNIVERSITY

Final Examination: Dec 13–16 2013
Course: COSC 2P90
Distributed: Fri. Dec 13, 5:00PM

Number of Pages: 17
Instructor(s): Earl Foxwell
Due: **Mon. Dec 16, 5:00PM**

Marks													
A		B		C		D		E		F		G	
1.	/3	8.	/4	13.	/6	19.	/2	24.	/2	29.	/7	31.	/12
2.	/3	9.	/3	14.	/2	20.	/3	25.	/2	30.	/4	32.	/3
3.	/2	10.	/4	15.	/3	21.	/3	26.	/5			33.	/3
4.	/3	11.	/3	16.	/8	22.	/2	27.	/4				
5.	/1	12.	/4	17.	/5	23.	/3	28.	/3				
6.	/2			18.	/4								
7.	/1												
	/15		/18		/28		/13		/16		/11		/18
Total:													/119

Read Carefully: A minimum of 40 percent must be obtained on this final examination in order to achieve a passing grade in the course. You must print out this exam, single-sided, and then answer all questions on that exam paper. It *must* be stapled, preferably with a departmental cover page bearing your student number, and properly signed. **When complete, submit this exam to the 2P90 dropbox. Absolutely no lates will be accepted; no exceptions!**

Answer *all* questions, unless otherwise noted. Feel free to use the backs of pages where necessary. All electronic communication with your instructor must be done via your *Brock* email account. For the duration of the exam, you are not permitted to discuss it with anyone except Earl Foxwell. This exam is **to be completed by yourself, alone**. You may not work with a partner, or *copy* answers from any source (see exceptions below). The following list represents the only exceptions to the rule that all work must be your own:

- The official course textbooks may be quoted without reference.
- The course slides from the 2013 Fall term of 2P90 may be quoted without reference.
- Any example I have provided you on the whiteboard or as source code for this session of 2P90 may be quoted without reference.
- Facts may be cited from other materials if properly referenced, but credit will only be awarded for ***demonstrated*** knowledge. Other students do not qualify as "other materials."
- Questions ending with a **(F)** tag reflect basic, common facts. There is no need to "make these your own."
- All answers must be unique to this submission; using a previous term's (or previous course's) material will qualify as academic misconduct.

To receive a grade for this exam, you *must* sign the following Declaration of Academic Integrity:

I, _____, have read the rules above, and assert that I have abided by them to the best of my knowledge. This work is my own.

Signature: _____ Student Number: _____

A. Short Answer

1. Answer all of the following: **3×1 (3)**

a) Make an argument for assembly language being either strongly-typed, weakly-typed, or untyped.

b) Make an argument for Java as a strongly-typed language.

c) Make an argument for Java as a weakly-typed language.

2. Concerning *Multiple Inheritance*:

a) What is it? **(F) (1)**

b) To what extent can we achieve it in Java? (To what extent *can't* we?) **(2)**

3. Concerning *aggregation* and *composition* in O-O, tell me what each is, including a simple example of each that makes the difference between them unambiguous. **(2)**

4. Considering inheritance: **3×1 (3)**
a) What is the difference between a derived class and a child class? **(F)**

b) What is the difference between *overloading* and *overriding*? **(F)**

c) When a child class *overrides* a method, can the parent's method still be used? If so, how? **(F)**

5. When we add the keyword `abstract` to method headers in Java interfaces, what does that keyword change? **(F) (1)**

6. *Briefly* explain two *different* reasons to use an *abstract class.* **2×1 (2)**

7. In the shortest answer you can give, what is *inheritance*? **(F) (1)**

B. Long Answer

8. For a first-year course, I could create a class, `GUIDealie`, to allow users to add *widgets* to a form, by extending `JFrame`, and obscuring most of the intricacies of implementation. i.e. I could make JFrames simpler to use by including easy-to-use methods in my `GUIDealie`. A side effect of this approach is that the `GUIDealie` could still do everything that a `JFrame` could do. **(4)**

a) Make an argument that this would be a good thing. **(1)**

b) Make an argument that this would be a bad thing. **(1)**

c) Assuming we decided that (b) outweighed (a), propose (and explain) a solution that utilizes some design aspect we've covered in class. **(2)**

9. Concerning Garbage and Garbage Collection: $\mathbf{3 \times 1}$ **(3)**

a) What *is* garbage? How is it created? **(F)**

b) Draw a diagram illustrating your answer for (a).

c) Why doesn't the garbage collector need to reclaim memory from the stack?

10. In class, we looked at memory safety (e.g. referencing data outside of declared boundaries). Give two examples of violating memory safety. (Assume C. Include code if you like) **2×2 (4)**

11. We've decided that *side effects* are frequently considered bad.
a) Explain why they can be bad, and include an example. Be thorough! **(2)**

b) In what circumstances/applications are they *especially* bad, and why? **(1)**

12. Consider *assertions* and *exceptions* in Java. **2×2 (4)**
a) What are exceptions, and what are they for?

b) Why is this both similar to, and different from, assertions?

C. Thought Exercises

13. Code? Data? Why not both? **3×2 (6)**

a) In C, we can simulate much of the functionality of classes by using structs and function pointers. What is a special (specific) benefit of using this approach?

b) Even though several dynamic languages like Python have objects anyway, we can *also* achieve a comparable result by applying the same basic technique as (a). Explain how.

c) Though both approaches (in *a* and *b*) are very powerful, they also have obvious potential problems. Identify and explain one of them.

14. I think I'd like a record that can hold a *colour* as three bytes (red, green, and blue). I remember that, in C, you use `unsigned char`s in place of the bytes, but there's still something wrong here.
I tried using it, and verifying on blue, as below, which checks out, but... I know it's broken!
Add one line of code that *proves* that my code is broken, and then fix it! **2×1 (2)**

```
typedef union colour {
   unsigned char red;
   unsigned char green;
   unsigned char blue;
} colour;
```

```
colour sky;
sky.red=100;
sky.green=200;
sky.blue=255;
printf("%i\n",sky.blue);
```

15. Javascript arrays have a built-in `sort()` function that arranges its elements into what it believes are ascending order. However, you can readily define an alternate sequence by providing an alternative sorting function.
For example, consider the following code:. **3×1 (3)**

```
function shorties(a, b) {
   return (b.length<a.length)?1:(b.length==a.length)?0:-1;
}
var arr=Array('flip','phone','zzt','field');
arr.sort(shorties);
```

a) Describe the ordering the `shorties` function actually imposes.

b) Rewrite the code to impose the *opposite* ordering.

c) What tool in Java is closest to this same functionality? (Answer carefully!)

16. Of the four programming paradigms — Imperative, Functional, Logic, and Object-Oriented — tell me why each is valuable, and tell me a problem or task for which each is uniquely suited. Note: Do *not* tell me *what* they are! **4×2 (8)**

17. Concerning the glory of Ada: **(5)**
a. Choose a *specific* problem that might (reasonably) occur with a common (popular) language other than Ada. **(1)**

b. Clearly explain how that problem might arise. **(2)**

c. Show why it wouldn't be a concern in Ada. **(2)**

18. Explain how either `Object` (or some other *supertype*), or *generics* (possibly bounded) might be used for the *same* problem. Demonstrate a benefit of either approach over the other. Include some Java code to illustrate. **(4)**

D. Functional

19. Concerning lambda functions in Lisp: **2×1 (2)**

a) What are they? **(F)**

b) Give a valid specific possible use for them.

20. Consider the following Lisp code: **(3)**

```
(defvar u '(bro))
(defun dealie() (format t "~a~%" u) (let ((u u)) (stuff))
      (format t "~a~%" u))
(defun stuff() (format t "~a~%" u) (push 'mad u) (format t "~a~%" u))
```

a) What is the output of invoking `(dealie)`? **(F) (1)**

b) Explain what's really going on to produce that output. **(2)**

21. If you have 5 or more monkeys, you're a happy person. If you have fewer than 5, then that's insufficient. On a side note, you like clean, well-coded solutions.
Consider the following solution for `report`ing on your current monkey status:

```
(defun output() (format t "You currently have ~d monkeys.~%" *monkeys*))
(defun inadequate() (format t "That's just not good enough.~%"))
(defun adequate() (format t "That's just spiffy!~%"))
(defun report(x) (defvar *monkeys* x) (output) (if (> x 4) (adequate) (inadequate)))
```

Is this a good solution? A bad solution? Prove your case, and be thorough! **(3)**

22. Consider the following Lisp function that is supposed to return whether or not a value is an odd number:

```
(defun checkOdd(value)
     (if (= (mod value 2) 1) (format t "Huzzah! You found an odd one!~%") T))
```

What is the problem with the code? Fix it. **(2)**

23. Mystery Program Repair! Lisp Edition **(3)**
Carefully examine the following program and sample invocation:

```
(defun schmindividual(canuck buzz)
   (let ((eh (abs canuck)) (bee (abs buzz)))
      (if (> bee eh) bee eh)))

(defun totesworks(maybe broken)
   (schmindividual maybe broken))

> (totesworks '(1 2 3) '(3 2 1))
```

Your task is to identify the intended functionality of the program, and then repair it.

E. Logic

24. In Prolog, we can use a neat trick with `asserts` that lets us initialize and maintain counters with arbitrary descriptors. If it worked, we could query `counter(Label,Count)` to check up on them. However... my attempt seems to be broken. **2×1 (2)**

a) *Fixxit!* (please?)

```
:- dynamic counter/2.
increment(Descriptor):-
        retract(counter(Descriptor,X))->
                (X = X+1, assert(counter(Descriptor,X)));
                (assert(counter(Descriptor,1))).
```

b) Write code that would use that code to make a counter, and increase it to 5.

25. Suppose we have the following Prolog facts loaded:

```
person(christopher). person(david). person(matt). happy(david). happy(matt).
```

Why, *exactly*, will `person(X),\+happy(X)` yield different results from `\+happy(X),person(X)`? **(2)**

26. Write Prolog rules and facts to reflect the following: **(5)**

- Kiwis have 2 legs.
- Ferrets and tripods have 3 legs.
- Lions, tigers, and bears have 4 legs.
- Animals with two legs are `bipedal`.
- Animals with four legs are `quadripedal`.
- Any other animals are `adorable`

Include a sample query to show all adorable animals. (Possession of legs constitutes animality)

27. Concerning cuts: **4×1 (4)**
a) What, *precisely*, do cuts actually do? **(F)**

b) What do we mean by a *green cut*? **(F)**

c) What do we mean by a *red cut*? **(F)**

d) Can red cuts be a *good* thing? Why or why not?

28. Mystery Program Repair! Prolog Edition **(3)**
Consider the following Prolog program:

```
wobble([],0).
wobble([_|Tip],Fries):-
    wobble(Tip,OnionRings),
    OnionRings is Fries+1

margarine(Dealie,Dealie).
margarine(Dealie,Stuff):-
    Stuff is Dealie+1.
margarine(Dealie,Stuff):-
    Dealie is Stuff+1.
```

```
spare([],End,End).
spare([A|P],End,[A|Pend]):-
    spare(P,End,Pend).

bowling(Pins,Seven,Ten):-
    spare(Seven,Ten,Pins),
    wobble(Seven,Homer),
    wobble(Ten,Apu),
    margarine(Homer,Apu),!.
```

Your task is to identify the intended functionality of the program, and then repair it.

F. UML

29. Choose one of the following tasks/environments: **(7)**

> A microwave oven
> An automobile
> A tabbed text editor
> Solitaire (the card game)
> A sewing machine
> A video-streaming client

- Indicate which you chose
- Write use cases for it **(3)**
- Draw a use case diagram for it **(1)**
- Draw UML class diagrams for it **(3)**

Note: include multiplicities and members (**including visibility**) for each class!

30. Show me UML class diagrams (including aggregation, composition, and inheritance, where appropriate) to show a suitable structure for the premise below. Ignore members methods and attributes and add additional classes where necessary. **(4)**

Robots can be arranged and configured to act as an intelligent *swarm*. Swarms are comprised of *agents* that act both independently and cooperatively. Each agent has some number of *effectors*, which are comprised of one or more *actuators* (e.g. motors). Each agent also has one or more *sensors*, which could be based on *light*, *sound*, or *touch*.

G. Design/Patterns

31. For each of the following design patterns: **4 × 3 (12)**

i. Give me a rough explanation of what it is. **(F) (1)**

ii. Explain a *specific* possible case where it would objectively improve design/usability. Be sufficiently detailed to prove your case! **(2)**

a) Adapter

b) Façade

c) Delegation

d) Observer

32. Tell me about the Model-View-Controller Pattern: **(3)**
a) Give a brief description. **(F) (1)**

b) Make a comprehensive argument extolling the virtues and power of the MVC pattern. **(2)**

33. Name a good (specific) case where an Abstract Factory could be helpful. Explain its contribution (include an explanation of how the problem would be addressed in an inferior way without it, and how the AF ameliorates the predicament). **(3)**

Bonus: If, in future iterations of this course, one had to choose to: a) still use javascript for the first assignment, but not in the form of a Chrome Extension; b) use python for some other task instead of javascript at all; c) still use javascript, but as an in-class component, rather than just in the assignment; d) leave it as it is; or e) something else entirely... which would you suggest? (If *e*, please suggest a 'something else')