CS131 Fall '23 - Final Exam

December 15th, 2023

Full Name (**LAST**, FIRST): _____

Student ID: _____

Signature: _____

Please be aware that there are **four** versions of this exam.
All cases of cheating will be reported to the Dean's office.

1. Mostly Old Stuff	/15
2. Bound to Get Passed Up	/8
3. Avoid Garbage Collection... Recycle Instead	/12
4. Is Your Template Too Generic?	/10
5. I Must Have Made an Error	/17
6. Are your objects classy?	/10
7. I Re-iterate...	/8
8. Prolog Scares Me	/18
Total	**/98**

Good luck!

1. Mostly Old Stuff (15 points)

a. (5 points) In our functional programming lectures we learned how to manually convert functions of multiple arguments into a curried form. Let's apply that approach to Python. Consider the following Python function:

```
# returns a function f(x,y) = c1*x^2 + c2*y + b
def gen_func(c1, c2, b):
  return lambda x, y: c1*x**2 + c2*y + b
```

If we had a curried version of *gen_func*, called *gen_func_c*, it could be used as follows:

```
q = gen_func_c(3)
r = q(5)
s = r(100)
# The above would be the same as:
# s = gen_func(3, 5, 100)

print(s(1,2))   # prints 113
```

Write a curried version of the *gen_func_c* function in Python here:

b. (5 points total) For this problem you have to deduce the properties of an unknown language. It is guaranteed that the language is either *statically typed* or *dynamically typed*, but not *gradually typed*.

i. (3 points) Consider the following code which is runs without errors:

```
class Animal {
 func eat() { ... }
};
class Dog : Animal { ... };
class Cat : Animal { ... };

func foo(var x) {
  x.eat();
  x = Cat("Felix");
}

func main() {
  var d = Dog("Koda");
  var c = Cat("Tiger");

  foo(d);
  foo(c);
}
```

Considering *just the code above*, is it *possible* for this language to be statically typed? If not, why not? If yes, what underlying language features or assumptions must exist to support this? Limit your answer to 3 sentences.

ii. (2 points) You're given some additional source code from the language, which runs and prints "woof" and "meow":

```
class Animal {
 func eat() { ... }
};
class Dog : Animal { ... };
class Cat : Animal { ... };

func foo(var a) {
  if (type_of(a) == Dog)
    a.bark();

  if (type_of(a) == Cat)
    a.meow();
}

func main() {
  var d = Dog("Koda");
  var c = Cat("Tiger");

  foo(d);
  foo(c);
}
```

Assuming the code above runs without error, can you conclude that this is a dynamically typed language, a statically typed language, or can we not tell? Explain your answer in 2-3 sentences.

c. (5 points total) Consider the following Haskell code:

```
data Outcome = ... -- outcome of operation
data Details = ... -- details if something went wrong

-- the following function computes an integer result if it's successful
compute :: Int -> Outcome
compute x =
    ... -- implementation not shown

processResult :: Int -> String
processResult x =
    case y of
        Success val -> "Output: " ++ show val
        Failure InvalidInput -> "Invalid input"
        Failure Overflow -> "Overflow"
        Failure Underflow -> "Underflow"
    where
        y = compute x
```

i. (1 point) Of the different types of error handling we learned in class, which type is being demonstrated by this code? Explain why in 2-3 sentences?

ii. (4 points) Show algebraic data type definitions in Haskell for the Details and Outcome types.

2. Bound to Get Passed Up (8 points)

Consider the following program written in a new language called Hopper. In Hopper, all variables and parameters have their binding semantics and parameter passing semantics specified explicitly when the variable/parameter is defined. Here are the options:

- val: value semantics
- ref: reference semantics
- objref: object reference semantics
- name: name semantics
- need: need semantics

You may assume arguments passed to functions are evaluated from left to right.

Given your understanding of these binding and parameter passing strategies, determine the output of the following Hopper program.

Write your answer here:

```
func bar(val a) {
  println("bar: ", a);
  return 2*a;
}

func foo(val a, name b, ref c) {
  println("foo");
  println(b);
  println(-b);
  c.radius = 3;
  c = Circle(radius:11);
}

func bletch(val a, need b, objref c) {
  println("bletch");
  println(b);
  println(-b);
  c.radius = 22;
  c = Circle(radius:33);
}

func main() {
  val c1 = Circle(radius:4);
  foo(bar(53), bar(5), c1);
  println("r: ", c1.radius);

  objref c2 = Circle(radius:4);
  bletch(bar(57), bar(9), c2);
  println("r: ", c2.radius);

  name x = bar(bar(11));
}
```

3. Avoid Garbage Collection… Recycle Instead (12 points)

Consider the following Java program which allocates a bunch of objects. Like C++, Java allocates objects via the **new keyword**.

```
class ClassA { };
class ClassB { };

class Class1 {
  public Class1() { this.ca = new ClassA(); }
  public ClassA get() { return this.ca; }
  private ClassA ca;
}

class Class2 {
  public Class2() { this.cb = new ClassB(); }
  private ClassB cb;
}

public class Main {
  // static methods are like class methods in Python
  public static void main(String[] args) {
    ClassB b = new ClassB();
    ClassA a = f1();
    f2(new ClassB());
  }

  private static ClassA f1() {
    Class1 c = new Class1();
    return c.get();
  }

  private static void f2(ClassB d) {
    Class2 e = new Class2();
    // Line A: execution pauses here for Garbage Collection
    ClassB f = new ClassB();
  }

  private static ClassB ss = new ClassB();    // ss is a class variable
}
```

a. (3 points) Assuming execution pauses on Line A for mark and sweep garbage collection, identify all of the root objects that would be identified by the garbage collection mark phase. You must identify each root object by the variable that refers to it.

b. (3 points) Assuming execution pauses on Line A for mark and sweep garbage collection, identify *all objects* that would be marked during the *full mark phase*. You may identify each marked object by the variable/member name that refers to it.

c. (2 points) Assuming execution pauses on Line A for mark and sweep garbage collection, identify all objects that would have been allocated, but not marked as in-use during the full mark phase. You may identify each unmarked object by the variable/member name that refers to it.

d. (2 points) You want to design a new version of the Brewin language for simulations where you expect to have a high rate of small objects allocated and deleted. Furthermore, while thousands of objects might be allocated and freed per second, the number of active (non collectable) objects at any one time will be O(100). Your goal is to minimize the amount of **computational overhead** performed by garbage collection across the execution of the program.

Of the three GC approaches we learned in class, which approach would you choose for Brewin? In 2-3 sentences, explain why.

e. (2 points) In CS33, you learned about virtual memory, specifically how RAM pages are swapped to disk to make room for other pages needed in RAM. This process, if it involves rapidly accessing many different pages over a short time span, can lead to 'thrashing,' where frequent paging in and out to disk significantly slows down the computer.

Assuming you have a program which initially allocates millions of active objects but, after initialization, only frees and allocates only a handful of objects per second, which of the three garbage collection methods covered in class is least likely to lead to thrashing during execution *after* initialization? Explain why in 2-3 sentences?

4. Is Your Template Too Generic? (10 points)

Natasha was given some code (written in an unknown language) in her Programming Languages class at USC that she thinks is a *generic class* designed to hold a bunch of items of a particular type:

```
type T = Any
class AnyHolder {
  private var values: List[T] = List()

  def addValue(newValue: T): Unit = {
    values = values :+ newValue  // :+ appends a new item to end of list
  }

  def getValue(index: Int): Option[T] = {
    if (index >= 0 && index < values.length) Some(values(index))
    else None
  }

  def printAllItems(): Unit = {
    // in this language, all objects guaranteed to have a toString method
    values.foreach(item => println(item.toString))
  }

  def getValues: List[T] = values
}
```

She said the example also came with the following code written in the same language:

```
class Circle(val radius: Double) {
  def area: Double = Math.PI * radius * radius
  def circumference: Double = 2 * Math.PI * radius
}

class Square(val side: Double) {
  def area: Double = side * side
  def perimeter: Double = 4 * side
}
```

And this code, shown on the following page:

```
// main function that creates a holder object and uses it
object Main extends App {

  var circleHolder = ... // code to define an object of type AnyHolder
  circleHolder.addValue(new Circle(5.0))

  var squareHolder = ... // code to define an object of type AnyHolder
  squareHolder.addValue(new Square(2.0))
}
```

After looking at the code, you're not so sure it's a generic. You remember from lecture that all classes in Java are implicitly derived from some base class called "Object" and you think that in this language, all classes might similarly be subclasses of the "Any" class.

a. (3 points) Using just the classes provided above, add code to the Main function below that can determine if the AnyHolder class is either (A) some type of generic/template class, or (B) a non-parametric class. Your code might compile and run for one case and result in a compile error for the other case.

```
// Usage example
object Main extends App {
  var holder = ... // code to define a holder object of type AnyHolder
  // Add your solution here









}
```

b. (2 points) Explain in 2-3 sentences how you would interpret the results of compiling/running your updated program to come to a conclusion on (A) vs. (B).

c. (2 points) Natasha is convinced that this class *is* using some type of parametric polymorphism, but isn't sure if it's a generic or a template. She's asked you if there's any way you could determine this. Explain in 2-3 sentences how you might update the AnyHolder class to determine whether it is a generic or a template.

d. (1 point) Assuming this class is NOT using parametric polymorphism and Natasha is wrong, what type of polymorphism, if any, is this class using when we insert objects like Circles and Squares?

e. (2 points) Assuming this code is NOT using parametric polymorphism and Natasha is wrong, what is the drawback of this approach vs. using templates or unbounded/bounded generics? Explain in 2-3 sentences.

5. I Must Have Made an Error (17 points)

Consider the following Java code which uses exceptions:

```
public class ExceptionDemo {
  // static methods are like class methods in Python
  public static void foo(int type) throws Exception {
    if (type == 1) throw new NullPointerException("NP");
    if (type == 2) throw new FileNotFoundException("FNF");
    if (type == 3) throw new ArrayIndexOutOfBoundsException("AIOOB");
    if (type == 4) throw new AssertionError("AE");
    System.out.println("None");
  }

  public static void foo1(int type) throws Exception {
    try {
      foo(type);
    }
    catch (IOException e) { System.out.println("foo1:IO");
                            return; }
    catch (NullPointerException e) { System.out.println("foo1:NP"); }
    finally { System.out.println("foo1:F"); }
  }

  public static void foo2(int type) throws Exception {
    try {
      create_temporary_file("foo.tmp");   // creates tmp file on disk
      foo1(type);
    }
    catch (RuntimeException e) { System.out.println("foo2:RT"); }
    finally { System.out.println("foo2:F"); }
  }

  public static void main(String[] args) throws Exception {
    try {
      foo2(0);  // Line A
      foo2(1);  // Line B
      foo2(2);  // Line C
      foo2(3);  // Line D
      foo2(4);  // Line E
    } catch (Throwable e) { System.out.println("main:T"); }
  }
}
```

<Problem continued on next page>

Next consider the Java exception hierarchy, which shows which exception classes are derived from which other exception base classes:

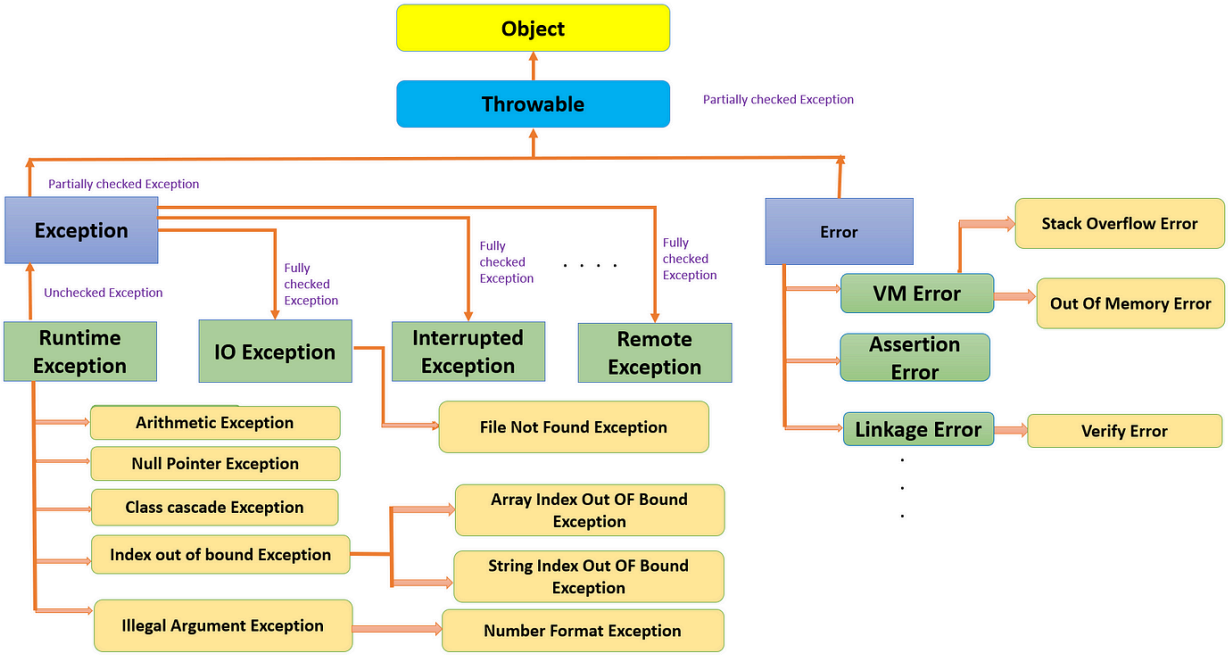

Fig. Exception Hierarchy in Java ~ by Deepti Swain

a. (3 points) What will this program print due to the call on line A?

b. (3 points) What will this program print due to the call on line B?

c. (3 points) What will this program print due to the call on line C?

d. (3 points) What will this program print due to the call on line D?

e. (3 points) What will this program print due to the call on line E?

f. (2 points) Of the three exception handling guarantees (*no-throw guarantee*, *strong exception guarantee* and *basic exception guarantee*), what is the strongest (most preferred) guarantee that function foo2() adheres to, if any? Why?

6. Are Your Objects Classy? (10 points)

a. (2 points) Is subtype polymorphism supported in dynamically-typed languages? If so, give an example. If not, explain why. Answer in three sentences or less:

b. (6 points) Consider the following program:

```
class Person {
public:
  void think() { ... }
  virtual void talk() {
    think();     // Line A
    cout << "Talking!\n";
  }
  virtual void eat() { ... }
};

class Student : public Person {
public:
  void study() {
    eat();       // Line B
    think();     // Line C
    cout << "Studying!\n";
  }
  virtual void talk() override { ... }
};

class HungryStudent : public Student {
public:
  virtual void eat() override { ... }
}
```

```
int main() {
  Student s;
  s.think();     // Line D

  Person *p = &s;
  p->think();    // Line E
  p->talk();     // Line F
}
```

<Continued on next page>

Determine whether dynamic or static dispatch is used on each line A-F. There's no need to explain why.

i. Is dynamic or static dispatch used on Line A?

ii. Is dynamic or static dispatch used on Line B?

iii. Is dynamic or static dispatch used on Line C?

iv. Is dynamic or static dispatch used on Line D?

v. Is dynamic or static dispatch used on Line E?

vi. Is dynamic or static dispatch used on Line F?

c. (2 points total) Joe has defined a base class in C++ called *SmartPerson*, and a second class called *Professor*, which is derived from SmartPerson using *implementation inheritance*.

```
class SmartPerson { ... };

class Professor: private SmartPerson { ... };
```

i. (1 point) What can we say about the typing relationship between SmartPerson and Professor? Use terms like subtype, supertype, etc. Answer in 2-3 sentences.

ii. (1 point) Consider the following code:

```
int main() {
  Professor p("Smallberg");
  std::vector<SmartPerson *> v;

  v.push_back(&p);   // Add pointer to Smallberg to end of vector
  std::cout << v[0]->get_name() << " says hi!" << endl;
}
```

What will the result be if we try to compile and run this program? Explain what happens and why in 2-3 sentences.

7. I Re-iterate… (8 points)

a. (5 points) Consider the following code which shows a binary search tree class in Python, with an *__iter__* method that returns a generator that performs pre-order traversal:

```
class Node:
  def __init__(self, key):
    self.left = None
    self.right = None
    self.val = key

class BinarySearchTree:
  def __init__(self):
    self.root = None

  def insert(self, key):
    # code not shown

  def __pre_order(self):
    stack = [self.root]

    while stack:
      node = stack.pop()
      if node:
        yield node.val
        stack.append(node.right)
        stack.append(node.left)

  def __iter__(self):
    return self.__pre_order()
```

```
# code to use our tree/generator
tree = BinarySearchTree()
tree.insert(5)
tree.insert(3)
tree.insert(7)
...

# Use the pre-order generator
for value in tree:
    print(value)   # prints 5 3 7
```

<Continued on next page>

Let's create an *iterator class* that performs a pre-order traversal on the BST. **Fill in the code for the *__next__* method to create a working iterator class which has the same functionality as the pre-order generator shown above. Your solution must be <= 10 lines:**

```
class PreOrderIterator:
    # passed in an object reference to the root node of the tree
    def __init__(self, root):
        self.stack = [root] if root else []

    def __next__(self):
      # you must add code to complete this method
```

b. (1 point) Now show how you'd modify your Python BST's *__iter__* method to use your iterator class:

```
class BinarySearchTree:
  ...

  def __iter__(self):
    # you must add code to complete this method
```

c. (2 points) Consider the following Ruby code which shows an enumerator (which is equivalent to a generator) that yields three numbers:

```
# Defines an enumerator (similar to a generator) that yields 1, 2 & 3
number_generator = Enumerator.new do |yielder|
  yielder << 1
  yielder << 2
  yielder << 3
end
```

In Ruby, you can use the enumerator as follows:

```
# Way #1 to use our enumerator:
puts number_generator.next  # Outputs: 1
puts number_generator.next  # Outputs: 2
puts number_generator.next  # Outputs: 3
```

And, surprisingly, you can also use the enumerator as follows:

```
# Way #2 to use our enumerator:
x = number_generator.each

puts x.next  # Outputs: 1
puts x.next  # Outputs: 2
puts x.next  # Outputs: 3
```

Using terms that we learned in class about iteration, describe what persona or personas a Ruby enumerator can act as. Limit your answer to 2-3 sentences.

8. Prolog Scares Me (18 points)

a. (8 points) Consider the following set of predicates which perform a mystery operation:

```
mystery(X, [], []).
mystery(X, [X|Y], [X|Q]) :- mystery(X, Y, Q).
mystery(X, [Y|Z], R) :- surprise(X,Y), mystery(X, Z, R).

surprise(P, Q) :- P \= Q.
```

i. (2 points) What will be the value of Answer if your run the following query (or explain why if you think the operation will result in an error, fail to unify, etc):

```
mystery(cat, [], Answer)
```

ii. (2 points) What will be the value of Answer if your run the following query (or explain why if you think the operation will result in an error, fail to unify, etc):

```
mystery(Answer, [mouse], [mouse])
```

iii. (2 points) What will be the value of Answer if your run the following query (or explain why if you think the operation will result in an error, fail to unify, etc):

```
mystery(cat, [dog, cat, cat], Answer)
```

iv. (2 points) What will be the value of Answer if your run the following query (or explain why if you think the operation will result in an error, fail to unify, etc):

```
mystery(lemur, [dog, lemur, dog, lemur], Answer)
```

b. (4 points) Given the following facts and rules:

```
parent(anish, benoit).    % anish is benoit's parent
parent(benoit, disha).
parent(anish, carlotta).
parent(benoit, eliza).
parent(carlotta, francesca).
parent(disha, emmy).

grandparent(Grandparent, Grandchild) :-
    parent(Grandparent, Parent),
    parent(Parent, Grandchild).
```

and your understanding of the algorithm Prolog uses for the Resolution process, what *exactly* will Prolog output for the following query:

```
grandparent(A, B)
```

What we're looking for here is not just the right pairs of items, but more importantly the *right ordering of those pairs*. Answering the right pairs in the wrong order gets 0 points.

Write your answer here:

c. (6 points) The *keep_unique* predicate is supposed to remove all duplicate items from a list (regardless of whether the duplicates are consecutive), leaving only unique items, e.g.:

keep_unique([], Answer) → Answer = [ ]
keep_unique([1,1, 2, 1], Answer) → Answer = [2, 1]
keep_unique([1, 2, 3, 1, 4, 2, 5], Answer) → Answer = [3, 1, 4, 2, 5]

where the resulting list may be in any order. **Fill in the blanks to ensure that these predicates properly implement the keep_unique predicate:**

```
keep_unique(__________, []).

keep_unique(____________, [X|Z]) :-
    not(member(X, ____________)),
    keep_unique(Y, ____________).

keep_unique([B|A], Q) :-
    member(B, ____________),
    keep_unique(____________, Q).
```

Hint: Prolog has a built-in predicate called *member* that is *true* if its first argument is a member of its second argument, which is a list: *member(a, [b,a,c]) → true*