CSCI 400 Practice Exam

Problem 1 (3)

Consider the following Java code:

```
public static void
    try {
        if (value != 10)
            throw new RuntimeException("Must be 10");
        System.out.println("Rin 10 10");
    } catch (Exception e) {
        System.out.println(e.getMessage());
    } finally {
        System.out.println("Cleaning up...");
    }
}
public static void main(String [] args) {
    mustBe10(10);
    mustBe5(5);
}
```

What will be displayed when it runs?

Problem 2 (2)

Consider the following Java code:

```
public class MyException extends Exception {
    public MyException(String msg) {
        super(msg);
    }
}
public void doSomething() {
    throw new MyException("This is bad!");
}
```

Would the compiler require a `throws` clause on the function `doSomething()`?

Problem 3 (4)

For the statements below, write 'J' if it applies only to Java, 'C' if it applies only to C++, 'B' if it applies to both.

_____ uses a `try/catch` syntax to handle errors.

_____ errors that are not handled are propagated to the caller.

_____ Can `throw` any data type.

_____ Can only `throw` objects that are a child of the `Throwable` class.

Problem 4 (1)

In C++, the statement:

```
int *nums = new int[5];
```

is an example of:

(a) Explicit heap dynamic lifetime
(b) Implicit heap dynamic lifetime
(c) Static lifetime
(d) Stack dynamic lifetime

Problem 5 (5)

One disadvantage of a language with only static lifetimes is that it would have poor support for recursion. Explain why this is the case. Which type of lifetime is better for recursion, and why?

Problem 6 (3)

Consider the following Ruby code:

```
def fun
  x = 5
end

puts fun
```

a. What would be displayed if this code is run?

b. If Ruby were based on statements rather than expressions, discuss how that might affect your answer to part **a**. Be specific, i.e. don't just say "it wouldn't work" – demonstrate that you really undertand what it means for Ruby to be based on expressions.

c. Which class of lifetime does `x` belong to?

(a) Explicit-heap dynamic lifetime
(b) Implicit-heap dynamic lifetime
(c) Static lifetime
(d) Stack dynamic lifetime

Problem 7 (1)

A constant pointer means you:

(a) Can't change where it points (i.e., the address stored in the variable)
(b) Can't change the content of the memory address it points to

Problem 8 (2)

Given the following line of Java code:

```java
Point p = new Point(3, 4);
```

Write a line of code that implicit dereferences `p`. Explain implicit dereferencing. You may assume the `Point` class has public variables `x` and `y` that are initialized by the constructor, which has the method signature `public Point(int x, int y)`.

Problem 9 (2)

Assume you have the following class:

```java
public class MyClass {

}
```

Write a single line inside of the class definition to show that Java is *not* strictly encapsulated.

Problem 10 (2)

Ruby's `attr_reader` is an example of metaprogramming. Use this example to explain metaprogramming.

Problem 11 (1)

Consider the following Java code:

```
public void doItThisWay(int i) { }
public void doItThisWay(String s) { }
```

Use this code to explain polymorphism.

Problem 12 (1)

True or **False** (Circle one): Duck typing provides support for polymorphism in Ruby.

Problem 13 (2)

Use the code below to explain duck typing. Name one main advantage of duck typing.

```
def pickLarger(x, y)
  if x > y
    puts x
  else
    puts y
  end
end
```

Problem 14 (2)

Consider the following Ruby code:

```
class MyMath
  def abs(n)
    n = -n if n < 1
  end
end
```

According to best practice, would it be better for `MyMath` to be a class or a module? Explain.

Problem 15 (3)

Let's say I just created a brand new object-oriented language. Inheritence is, of course, a big part of that language. Here's some code in that language:

```
class Parent {
    func doSomething() {
        print("Rabbit season!");
    }
}
subclass Child is-a Parent {
    func doSomething() {
        print("Duck season!");
    }
}
func main() {
    // the following line declares a parent variable but
    // initializes it to a child
    Parent p = new Child();
    // this line calls the `doSomething()` method
    p.doSomething();
}
```

Answer the following questions:

a. What will be displayed when the `main` method executes if the language uses late/dynamic binding?

b. What will be displayed when the `main` method executes if the language uses early/static binding?

c. Describe on advantage of early binding.

Problem 16 (1)

True or **False** (Circle one): In Ruby, if you want to add a function to the existing `Integer` class you would need to have access to the `Integer` class's source code.

Problem 17 (2)

A Singleton `method` in Ruby (circle all that apply):

(a) Applies only to one instance of a class.
(b) Ensures there's only one object of that type.
(c) Potentially reduces the need to create subclasses.

(d) None of the above.

Problem 18 (1)

Why do most modern programming languages have some type of “namespace” capability? Be specific.

Problem 19 (3)

The figure below illustrates the diamond problem as it pertains to C++.

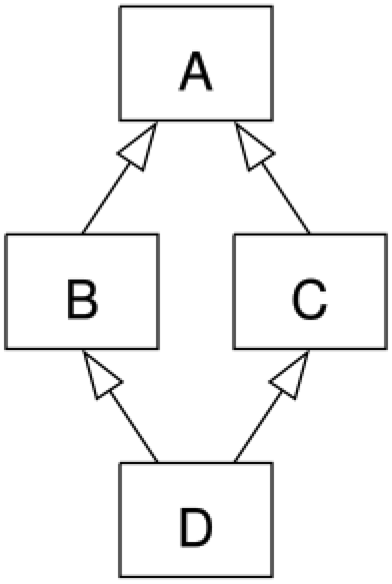

Figure 1: Diamond Problem

a. Use this figure to explain the diamond problem.

b. How do mixins solve this problem?

c. **True** or **False** (Circle one): Interfaces in Java solve the same problem.

Problem 20 (1)

Assume you’re writing a website that requires customers to enter their account numbers. Account numbers have the format:

- 2-digit number, represents a month (e.g. 01, 02, . . . , 11, 12)

- 2 capital letters, representing initials
- Optionally ends with one single digit

Examples of valid account numbers: 01CR, 12FA, 09JJ, 09JJ2.

Circle the regular expression that would accept/validate account numbers in this format.

(a) `[01][0-9][A-Z][A-Z]\d?`
(b) `(0[0-9]|1[0-2])[A-Z][A-Z]\d?`
(c) `(0[0-9]|1[0-2])[A-Z][A-Z]\d*`

Problem 21 (2)

Instance variables in Ruby are strictly encapsulated, **but** reflection breaks that encapsulation.

a. What is reflection?

b. Explain how reflection breaks encapsulation. Include an example (you don't need exact syntax).

Problem 22 (4)

The following Ruby functions do not follow the DRY principle:

```
def doIt(limit)
  limit.times { |i|
    puts i * 10
  end
end

def doIt2(limit)
  limit.times { |i|
    puts i * 2
  end
end

doIt 4
doIt2 4
```

Convert this code into one function named `doIt3` that makes use of `yield`. Include function calls to `doIt3` that would result in the same output as the two function calls above.