

COMP 333 Practice Exam #2

This is representative of the kinds of topics and kind of questions you may be asked on the midterm.

Prototype-Based Inheritance in JavaScript

1.a.) Define a constructor for Dog objects, where each Dog object has a name. An example code snippet is below, illustrating usage:

```
let d = new Dog("Rover"); // line 1
console.log(d.name);      // line 2; prints Rover
```

1.b.) Define a different constructor for Dog, which puts a bark method **directly** on the Dog objects. The bark method should print "Woof!" when called. Example usage is below:

```
let d = new Dog("Sparky");
d.bark(); // prints Woof!
```

1.c.) Define a method named growl for Dog objects, which prints "[dog name] growls" when called. Use Dog's **prototype**, instead of putting the method directly on Dog objects themselves. Example usage is below:

```
let d = new Dog("Rocky");
d.growl(); // prints Rocky growls
```

2.) Consider the JavaScript code below:

```
function Animal(name) { this.name = name; }
Animal.prototype.getName = function() { return this.name; }
function Bird(name) { Animal.call(this, name); }
Bird.prototype = Object.create(Animal.prototype);
Bird.prototype.fly = function() {
  console.log(this.getName() + " flies");
}
function Mouse(name) {
  this.name = name;
  this.squeak = function() {
    console.log(this.name + " squeaks");
  }
}
Mouse.prototype = Object.create(Animal.prototype);
Mouse.prototype.fly = Bird.prototype.fly;
let b1 = new Bird("Coco"); let b2 = new Bird("Sunny");
let m1 = new Mouse("Pip"); let m2 = new Mouse("Ruby");
```

Write a memory diagram which shows how memory looks after this program executes. Your diagram should include the objects and fields associated with `b1`, `b2`, `m1`, `m2`, `Mouse.prototype`, and `Bird.prototype`, `Animal.prototype`. As a hint, the `__proto__` field on objects refers to the corresponding object's prototype.

3.) Consider the test suite below, using `assertEquals` from the second assignment:

```
function test1() {
  let t1 = new Obj("foo");
  assertEquals("foo", t1.field);
}

function test2() {
  let t2 = new Obj("bar");
  assertEquals("barbar", t2.doubleField());
}

function test3() {
  let t3 = new Obj("baz");
  // hasOwnProperty returns true if the object itself has the field,
  // otherwise it returns false. If the field is on the object's
  // prototype instead (__proto__), it returns false.
  assertEquals(false, t3.hasOwnProperty("doubleField"));
}
```

Write JavaScript code which will make the above tests pass.

Language Design / Implementation Concepts

4.) Languages which compile to machine code tend to run faster than languages which are interpreted. Provide one possible reason why.

5.) FooBar is a program with a C version (compiled) and a Java version (interpreted). The C and Java versions work identically and are implemented as similarly as possible.

5.a.) When FooBar is run for only a short period of time, the C version always runs faster than the Java version. Explain why this might be.

5.b.) When FooBar is run for a long amount of time, the Java version always runs faster than the C version. Explain why this might be.

6.) You've been tasked to write control software for a satellite that will orbit the Earth. The software you write practically must be correct, as it will be exceedingly difficult to remotely fix bugs or reset the system. The satellite's hardware is considered very underpowered, with a processor design which is slow and multiple decades old, and a relatively small amount of available RAM. You've been given total control over the language you'll write this software in, and you've narrowed your choice of language down to C and Java.

6.a.) Provide one reason why you should use C, and one reason why you shouldn't use C.

6.a.) Provide one reason why you should use Java, and one reason why you shouldn't use Java.

7.) Both reference counting and garbage collection are systems used to automatically reclaim memory that is no longer in use. However, reference counted systems may additionally have a garbage collector available (e.g., Python). Why would a reference-counted system additionally need a garbage collector?

Pattern Matching in Swift

(Note: this might not be on the exam, depending on how far we get Tuesday.)

8.) Consider the following `enum` definition:

```
enum SomeEnum {
    case foo(Int)
    case bar(Int, Int)
    case baz(Int, Int, Int)
}
```

Write a function named `test` which takes a value of type `SomeEnum`. The function should do the following:

- If given a `foo`, it should return the value in the `foo`
- If given a `bar`, it should return the sum of the two values in the `bar`
- If given a `baz`, it should return the sum of the **first** and **last** values in the `baz`. You should **not** introduce a variable for the second (middle) value in the `baz`.

An example call to the function follows: `test(SomeEnum.baz(1, 2, 3))`

Generics and Higher-Order Functions in Swift

(Note: this might not be on the exam, depending on how far we get Tuesday.)

9.) Write the body of the following function, or say if it's impossible to implement. If it's impossible to implement, explain why.

```
func combine<A, B>(a: A, b: B) -> (A, B) {  
  
  
  
  
  
  
  
  
  
}
```

10.) Write the body of the following function, or say if it's impossible to implement. If it's impossible to implement, explain why.

```
func combine2<A, B>(a: A) -> ((B) -> (A, B)) {  
  
  
  
  
  
  
  
  
  
}
```

11.) Write the body of the following function, or say if it's impossible to implement. If it's impossible to implement, explain why.

```
func combine3<A, B>(tup: (A, B)) -> A {  
  
  
  
  
  
  
  
  
  
}
```

12.) Write the body of the following function, or say if it's impossible to implement. If it's impossible to implement, explain why.

```
func combine4<A, B>(a: A, f: (A) -> B) -> (A, B) {  
  
  
  
  
  
  
  
  
  
}
```

13.) Consider the following `enum` definition:

```
enum Something<A, B, C> {  
  case alpha(A)  
  case beta(B)  
  case gamma(C)  
}
```

13.a.) Write the body of the following function, or say if it's impossible to implement. If it's impossible to implement, explain why.

```
func combine5<A, B, C>(s: Something<A, B, C>) -> (A, B, C) {
```

```
}
```

13.b.) Write the body of the following function, or say if it's impossible to implement. If it's impossible to implement, explain why.

```
func combine6<A>(s: Something<A, A, A>) -> A {
```

```
}
```

14.) Write the body of the following function, or say if it's impossible to implement. If it's impossible to implement, explain why.

```
func combine7<A, B>(f: (A) -> B, b: B) -> A {
```

```
}
```