**COMP 333 Practice Exam**

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

**Virtual Dispatch in Java**

1.) Consider the following Java code:

```
public interface I1 {
  public void doThing();
}
public class C1 implements I1 {
  public void doThing() { System.out.println("c1"); }
}
public class C2 implements I1 {
  public void doThing() { System.out.println("c2"); }
}
public class Main {
  public void makeCall(I1 value) {
    value.doThing();
  }
  public static void main(String[] args) {
    I1 t1 = new C1();
    I1 t2 = new C2();
    makeCall(t1);
    makeCall(t2);
  }
}
```

What is the output of the `main` method above?

2.) Consider the following code snippet:

```
public class Main {
  public static void main(String[] args) {
    Operation op1 = new AddOperation();
    Operation op2 = new SubtractOperation();
    int res1 = op1.doOp(5, 3);
    int res2 = op2.doOp(5, 3);
    System.out.println(res1);
    System.out.pritnln(res2);
  }
}
```

Define any interfaces and/or classes necessary to make this snippet print 8, followed by 2.

**Prototype-Based Inheritance in JavaScript**

3.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");
console.log(d.name); // prints Rover
```

3.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!
```

3.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
```

4.) 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.

5.) Consider the test suite below, using `assertEquals` from the first 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");
  assertEquals(false, t3.hasOwnProperty("doubleField"));
}
```

Write JavaScript code which will make the above tests pass.

**Higher-Order Functions in JavaScript**

6.) Write the output of the following JavaScript code:

```
function foo(fooParam) {
  return function (innerParam) {
    return fooParam - innerParam;
  }
}

let f1 = foo(7);
let f2 = foo(10);
console.log(f1(2));
console.log(f2(3));
console.log(f1(4));
console.log(f2(5));
```

7.) Write the output of the following JavaScript code:

```
function guard(thing) {
  try {
    return thing();
  } catch (error) {
    return "ERROR";
  }
}

function f() {
  throw "hello";
}

console.log(guard(f));
console.log(guard(function() { return 42; }));
```

8.) Consider the following array definition in JavaScript:

```
let arr = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

8.a) Use `filter` to get an array of all even elements in `arr`.

8.b) Use `map` to get an array of strings, where each string represents a number in `arr`. As a hint, you can call the `toString()` method on a number (e.g., `5.toString()`) in JavaScript to get its string representation.

8.c) Use `reduce` to get the last element in `arr`.

8.d) Use a combination of `filter` and `reduce` to get the sum of all elements in `arr` which are greater than `5`.

**Algebraic Data Structures in Swift**

9.) Consider the following information:

- A `TrafficDevice` is either a `stopSign` or a `trafficLight`. A `trafficLight` is associated with a specific `LightColor`.
- A `LightColor` can be one of `red`, `yellow`, or `green`.

9.a.) Write two `enum` definitions in Swift which represent this information.

9.b.) Define a **mutable** variable named `lc` which holds the `red` color. The type of the variable should be `LightColor`.

9.c.) Define an **immutable** variable named `td` which holds a traffic light with the `green` color. The type of the variable should be `TrafficDevice`.