COMP 333
Summer 2025
Midterm Practice Exam #1

This is representative of the kinds of topics and kind of questions you may be asked on the midterm. This practice exam, along with assignment 1 and the in-class handout on Java, are intended to be comprehensive of everything on the exam. That is, I will not ask anything that's not somehow covered by those sources.

You are permitted to bring two 8.5 x 11 sheets of paper into the exam with you, as long as they have handwritten notes on them. Both sides of both sheets can be used. To be clear, these must be entirely handwritten.

Virtual Dispatch - Conceptual Understanding

1.) Name one reason why someone might want to use virtual dispatch (i.e., Java's way of implementing ad-hoc polymorphism).

2.) Name one reason why someone might **not** want to use virtual dispatch (i.e., Java's way of implementing ad-hoc polymorphism).

Virtual Dispatch in Java

3.) 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?

4.) Consider the following code snippet:

```
public class Main {
  public static void main(String[] args) {
    Operation op1 = new AddOperation();      // line 3
    Operation op2 = new SubtractOperation(); // line 4
    int res1 = op1.doOp(5, 3);               // line 5
    int res2 = op2.doOp(5, 3);               // line 6
    System.out.println(res1); // line 7; should print 8
    System.out.pritnln(res2); // line 8; should print 2
  }
}
```

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

5.) Consider the following incomplete Java code and output:

```
public class Incomplete {
  public static void printResult(final Runner r, final int i) {
    boolean result = r.someMethod(i);
    System.out.println(result);
  }
  public static void main(final String[] args) {
    final IsEven even = new IsEven();
    printResult(even, 3); // prints false
    printResult(even, 4); // prints true
    final IsLessThan ltFive = new IsLessThan(5);
    printResult(ltFive, 4); // prints true
    printResult(ltFive, 6); // prints false
    final IsLessThan ltZero = new IsLessThan(0);
    printResult(ltZero, -1); // prints true
    printResult(ltZero, 1); // prints false
  }
}
```

Define any interfaces and/or classes necessary to make the output in the comments work. You should not have to modify any code here. Multiple answers are possible.

6.) Consider the following Java code, which simulates a lock which can be either locked or unlocked. The lock is an immutable data structure, so locking or unlocking returns a new lock in an appropriate state.

```java
public class Lock {
    private final boolean locked;

    public Lock(final boolean locked) {
        this.locked = locked;
    }

    public Lock unlock() {
        if (locked) {
            System.out.println("lock unlocked");
            return new Lock(false);
        } else {
            System.out.println("lock already unlocked");
            return this;
        }
    }

    public Lock lock() {
        if (!locked) {
            System.out.println("lock locked");
            return new Lock(true);
        } else {
            System.out.println("lock already locked");
            return this;
        }
    }

    public boolean isLocked() {
        return locked;
    }
}
```

Refactor this code to use virtual dispatch, instead of using `if`/`else`. As a hint, you should have a base class/interface for `Lock`, and subclasses for locked and unlocked locks. `Lock` itself doesn't need a constructor, and you do not need to worry about maintaining compatibility with existing code that uses `Lock`.

---(Continued on to next page)---

Types

7.) The code below does not compile. Why?

```
public interface MyInterface {
  public void foo();
}

public class MyClass implements MyInterface {
  public void foo() {}
  public void bar() {}

  public static void main(String[] args) {
    MyInterface a = new MyClass();
    a.bar();
  }
}
```

8.) Java supports subtyping polymorphism. Write a Java code snippet that compiles and uses subtyping polymorphism.

9.) Name one reason why someone might prefer static typing over dynamic typing.

10.) Name one reason why someone might prefer dynamic typing over static typing.

11.) Name one reason why someone might prefer strong typing over weak typing.

12.) Name one reason why someone might prefer weak typing over strong typing.

13.) Consider the following code, written in some unknown language:

```
define doSomething(x) {
  return x.foo(7);
}

obj = Foo("hello")
value = doSomething(obj)
print(value)
obj = Foo("goodbye")
```

Provide an argument why this language might be statically-typed, OR why it might be dynamically-typed. Both are possible; the explanation why is the only important part.

14.) Consider the following code snippet in some unknown object-oriented language:

```
Object obj = new Foo();
Foo f = (Foo)obj;
```

The following checks are relevant:

1. Ensure that class `Object` and `Foo` exist
2. Ensure that `Foo`'s constructor takes no arguments
3. Ensure that `obj` is in scope on the second line
4. With respect to the cast on the second line, ensure that `obj`'s type is actually `Foo`

Identify which checks are most likely to be performed when for each kind of language.

	Statically-Typed	**Dynamically-Typed**
Strongly-Typed	Runtime: Compile-time:	Runtime: Compile-time:
Weakly-Typed	Runtime: Compile-time:	■

Higher-Order Functions in JavaScript

15.) 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));
```

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

```
function cap(min, max, wrapped) {
  return function (param) {
    let temp = wrapped(param);
    if (temp < min) {
      return min;
    } else if (temp > max) {
      return max;
    } else {
      return temp;
    }
  };
}

function addTen(param) {
  return param + 10;
}

function subTen(param) {
  return param - 10;
}

let f1 = cap(0, 10, addTen);
let f2 = cap(0, 100, addTen);
let f3 = cap(0, 10, subTen);
let f4 = cap(0, 100, subTen);

console.log(f1(0));
console.log(f1(5));
console.log(); // prints an empty line

console.log(f2(0));
console.log(f2(5));
console.log(); // prints an empty line

console.log(f3(0));
console.log(f3(5));
console.log(); // prints an empty line

console.log(f4(0));
console.log(f4(5));
console.log(); // prints an empty line
```