There are 6 problems on this exam. It is 16 pages long, so make sure you have the whole exam. You will have 150 minutes in which to work on the problems. You will likely find some problems easier than others; read all problems before beginning to work, and use your time wisely. The examination is worth 100 points total (1.5 minute/point). The point breakdown for the parts of each problem is printed with the problem. Some of the problems have several parts, so make sure you do all of them.

This is an open-book, open-notes examination. No electronic assistance is allowed.

When writing code, do not worry too much about getting the syntax exactly right. You will not be penalized unless it makes the meaning unclear. Use common sense: I do not expect you to write 100s of lines of code or text; if you find yourself doing so, there is a simpler, shorter answer.

Do all written work on the exam itself. If you are running low on space, write on the back of the exam sheets and be sure to write (OVER) on the front side. It is to your advantage to show your work—we will award partial credit for incorrect solutions that are headed in the right direction. If you feel rushed, try to write a brief statement that captures key ideas relevant to the solution of the problem. If you show your work, clearly indicate what your answer is.

If you finish in the last ten minutes of the exam, please remain in your seat until the end of the exam as a courtesy to your fellow students.

Name

Student ID
<table>
<thead>
<tr>
<th>Problem</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
1. True/False [20 pts]

(parts a–j; -1 point for each wrong answer, 0 points for each blank answer, 2 points for each correct answer. Therefore, the score for this problem is \( \max(0, 2 \cdot \#\text{correct} - \#\text{incorrect}) \).

a. ___ When making a tail recursive call, a new frame for the callee function need not be pushed onto the stack.

b. ___ Functional languages allow functions to be passed into and returned from other functions.

c. ___ If a variable is declared with type \( T \), any supertype of \( T \) can be assigned into that variable.

d. ___ With call-by-name semantics, the argument to a function is evaluated before the function call.

e. ___ Using synchronized blocks in Java ensures a multi-threaded program will never deadlock.

f. ___ Functional languages favor binding of values to variables over using assignment.

g. ___ A type-safe language ensures that a values are never used at an inappropriate type.

h. ___ In class-based object-oriented languages, objects are initialized with constructors.

i. ___ Two actors can access a shared variable as long as they acquire a lock.

j. ___ A supertype must support more operations than any of its subtypes.
2. Functional languages [20 pts]

Consider the following Scala functions on lists:

```scala
def foldLeft[A,B](z: B)(f: (B,A) => B)(xs: List[A]): B = xs match {
  case Nil => z
  case y::ys => foldLeft(f(z, y))(f)(ys)
}

def foldRight[A,B](z: B)(f: (A,B) => B)(xs: List[A]): B = xs match {
  case Nil => z
  case y::ys => f(y, foldRight(z)(f)(ys))
}
```

Recall that `Nil` is the empty list, and `::` is the “cons” operator that produces a new list by prepending the first argument to the second. For instance, `1::2::3::Nil` is the list `List(1, 2, 3)`

(a) [5 pts] Is the `foldLeft` function tail recursive? Explain why or why not.

(b) [5 pts] Is the `foldRight` function tail recursive? Explain why or why not.
(c) [5 pts] Write a call to the `foldLeft` function that returns the number of elements of a list `zs` of Int. Keep it functional—do not use any assignments.

(d) [5 pts] Write a call to either `foldLeft` or `foldRight` (one is easier than the other) that takes a list `zs` of Int and returns the reverse of the list. That is if `zs` is `List(1,2,3)`, the result should be `List(3,2,1)`. Keep it functional—do not use any assignments.
3. Continuations [10 pts]

Convert the following functions into continuation-passing style:

(a) [5 pts]

```python
def f(n: Int): Int = {
    if (n == 0)
        return 1
    else
        return 1 + f(n-1)
}
```
(b) [5 pts]

```scala
def find(n: Int, lo: Int, hi: Int, a: Array[Int]): Int = {
  if (lo == hi)
    return lo
  val mid = (lo + hi) / 2
  if (n < a(mid))
    return find(n, lo, mid, a)
  else
    return find(n, mid, hi, a)
}
```
4. OO languages [20 pts]

   (a) [4 pts] C++ supports multiple inheritance of arbitrary classes. Java only supports multiple inheritance of interfaces. List two reasons why.

   (b) [7 pts] Consider the following declarations:

```java
class Animal {
    void eat(Plant p) { System.out.println("Mmm, veges!"); }
    void eat(Animal a) { System.out.println("Mmm, meat!"); }
}
class Marsupial extends Animal {
    void eat(Animal a) { System.out.println("Meh."); }
}
class Koala extends Marsupial {
    void eat(Eucalyptus e) { System.out.println("Nom nom nom"); }
    void eat(Plant p) { System.out.println("Uh-oh"); }
    void eat(Animal a) { System.out.println("No thanks, I'm a vegan."); }
}
class Cow extends Animal {
}
class Plant {
}
class Eucalyptus extends Plant {

Koala koala = new Koala();
Marsupial mars = koala;
Animal steak = new Cow();
Eucalyptus euc = new Eucalyptus();
Plant plant = euc;
```
What is printed by executing the following Java code?

```java
mars.eat(plant);

mars.eat(euc);

koala.eat(plant);

koala.eat(euc);

koala.eat(mars);

koala.eat(steak);
```
(c) [5 pts] Consider the following program:

class C {
    int x;
    public boolean equals(Object o) {
        return o instanceof C && ((C) o).x == x;
    }
}
class D extends C {
    int y;
    public boolean equals(Object o) {
        return o instanceof D && super.equals(o) && ((D) o).y == y;
    }
}

Is the implementation of equals correct? If so, explain why.
If not, write code that demonstrates its incorrect behavior and then write corrected versions of equals for C and D.
(d) [4 pts] Consider the following generic classes in Java:

```java
class A { int a; }
class B extends A { int b; }
class C<T> { ... }
```

Then, I write the following:

```java
C<B> ob = new C<B>();
C<A> oa = ob;
...
```

Java will reject this program because it won’t type-check the assignment to `oa`. If you think it is unsafe to run this program, fill in the missing code to show how this can lead to a run-time type error.
5. Concurrency [15 pts]

Consider the following Java code:

```java
class C {
    int x;
    public int get() {
        return x;
    }
    public void inc() {
        x = x + 1;
    }
}
class R implements Runnable {
    C c1;
    C c2;
    R(C c1, C c2) { this.c1 = c1; this.c2 = c2; }
    public void run() {
        ...
    }
}
public class M {
    public static void main(String[] args) {
        C c1 = new C();
        C c2 = new C();
        Thread t1 = new Thread(new R(c1, c2));
        Thread t2 = new Thread(new R(c2, c1));
        t1.start();
        t2.start();
    }
}
```
(a) [5 pts] Fill in the body of `R.run` so that the program has a data race or storage conflict. Show an interleaving of reads and writes in the two threads `t1` and `t2` that exhibits the race. Show the values being read or written in each operation.

(b) [5 pts] Correct the implementation of `C` so that there is no race.
(c) [5 pts] Consider the following alternate implementation of `R.run`:

```java
public void run() {
    while (c1.get() == 0)
        c2.inc();
}
```

Using the original version of C above, is it guaranteed that both threads will eventually terminate? Explain why or why not.
6. Types [15 pts]

(a) [2 pts] Suppose we introduce range types to a language like Java. The type $[n_1, n_2]$ represents all int values between the integer constants $n_1$ and $n_2$, inclusive. For example, $[1,3]$ is the type of ints 1, 2, and 3.

Should $[n_1, n_2]$ be a subtype of int?

(b) [5 pts] When should $[n_1, n_2]$ be a subtype of another range type $[m_1, m_2]$? Be precise.
(c) [5 pts] Suppose we loosen the restriction that $m$ and $n$ be compile-time constants. That is, suppose we allow the type $[e_1, e_2]$ where $e_1$ and $e_2$ are arbitrary Java expressions of type int. Write a legal (according to the relaxed rules) sequence of statements in which a variable of a range type contains a value outside the range.

(d) [3 pts] Java distinguishes between reference types like Object and String and primitive types like boolean and long. Explain why.