

Name: _____

**CSE331 Autumn 2019, Final Exam
December 9, 2019**

Please do not turn the page until 8:30.

Rules:

- After the exam starts, rip out the last page and do not turn it in.
- The exam is closed book, closed notes, closed electronics, closed mouth, open mind.
- **Please stop promptly at 10:20.**
- There are **135 (not 100) points**, distributed **unevenly** among **9** questions (all with multiple parts):
- **The exam is printed double-sided.**

Advice:

- Read questions carefully. Understand a question before you start writing.
- Write down thoughts and intermediate steps so you can get partial credit. But clearly indicate what is your final answer.
- The questions are not necessarily in order of difficulty. Skip around. Make sure you get to all the questions.
- If you have questions, ask.
- Relax. You are here to learn.

Name: _____

1. (16 points) Like many problems in this exam, this problem refers to the `Range` class implemented on the last page of the exam. This problem focuses on the specification and implementation of the class itself.
 - (a) Specifying the ADT that `Range` implements would include a *class overview*. Give such an overview, including a definition of abstract values.
 - (b) Give an *abstraction function* for `Range`. Assume there are no ill-formed instances of `Range`, meaning the representation invariant always holds because it is just “true.”
 - (c) Give a *method specification* for `setLow`.
 - (d) Give a *method specification* for `toArray`, including a `@requires` clause that is strong enough to prevent `toArray` from throwing an exception, but not any stronger.

Name: _____

2. (7 points) Consider this code, which uses the `Range` class:

```
class CountDivisibleBy {
    private int count;
    private int divisor;
    public CountDivisibleBy(int d) {
        count = 0;
        divisor = d;
    }
    public void m(int i) {
        if(i % divisor == 0) {
            count++;
        }
    }
    public int getCount() {
        return count;
    }
}
class Main {
    public static void foo() {
        Range r = new Range(3,10);
        CountDivisibleBy it1 = new CountDivisibleBy(4);
        CountDivisibleBy it2 = new CountDivisibleBy(4);

        r.forEach(it1);
        System.out.println(it1.getCount());

        r.setHigh(17);

        r.forEach(it1);
        System.out.println(it1.getCount());

        r.forEach(it2);
        System.out.println(it2.getCount());
    }
}
```

- (a) The code does not type-check. Explain exactly how to change the code so that it does. (Your answer should be minimal, not “silly” answers like deleting almost everything.)
- (b) Assuming your fix from part (a), what does the `foo` method print when executed?

Name: _____

3. (9 points) This problem considers 3 variants of the `Range` class that are “fixed” (unchanging) because they do not have setter methods. Assume the `Range` ADT is specified according to your answers in Problem 1.

```
class FixedRange1 extends Range {
    public FixedRange1(int lo, int hi) {
        super(lo,hi);
    }
    public void setLow(int lo) { /* do nothing */ }
    public void setHigh(int hi) { /* do nothing */ }
}
class FixedRange2 {
    private int low;
    private int high;

    public FixedRange2(int lo, int hi) {
        low = lo;
        high = hi;
    }
    public int getLow() { return low; }
    public int getHigh() { return high; }
    public int[] toArray() {
        int[] ans = new int[high-low+1];
        for(int i=0; i < high-low+1; i++) {
            ans[i] = low+i;
        }
        return ans;
    }
    public void forEach(IntTaker it) {
        for(int i=low; i <= high; i++) {
            it.m(i);
        }
    }
}
class FixedRange3 {
    private Range r;

    public FixedRange3(int lo, int hi) {
        r = new Range(lo,hi);
    }
    public int getLow() { return r.getLow(); }
    public int getHigh() { return r.getHigh(); }
    public void forEach(IntTaker it) { r.forEach(it); }
    public int[] toArray() { return r.toArray(); }
}
```

- (a) Is `FixedRange1` a Java subtype of `Range`?
- (b) Is `FixedRange1` a true subtype of `Range`?
- (c) Is `FixedRange2` a Java subtype of `Range`?
- (d) Is `FixedRange2` a true subtype of `Range`?
- (e) Is `FixedRange3` a Java subtype of `Range`?
- (f) Is `FixedRange3` a true subtype of `Range`?

Name: _____

5. (15 points) In this problem, we create a subclass of the original `Range` class that has listeners that are notified whenever a bound is changed. Here is a partial solution:

```
interface BoundChangeListener {
    void onBoundChange(int low, int high);
}
class RangeWithBoundChangeListeners extends Range {
    private List<BoundChangeListener> listeners;
    public RangeWithBoundChangeListeners(int low, int high) {
        super(low,high);
        listeners = new ArrayList<BoundChangeListener>();
    }
    private void notifyListeners() { ... }
    public void addListener(BoundChangeListener listener) { ... }
}
```

- (a) Complete the implementations of `notifyListeners` and `addListener`.

- (b) Complete the implementation by overriding methods as needed to notify listeners of bounds changes.

Parts (c) and (d) are on the next page.

Name: _____

(c) Consider:

```
class MysteryListener implements BoundChangeListener {
    public void onBoundChange(int low, int high) {
        throw new Error("I did not want that to happen");
    }
}
```

How would adding an instance of `MysteryListener` to an instance of `RangeWithBoundChangeListeners` affect the behavior of the `RangeWithBoundChangeListeners`?

- (d) Suppose we want to have a listener count how many times the high bound of a range is *increased*, meaning the new bound is higher than the old bound.
- i. Why is implementing such a listener difficult or impossible?
 - ii. Describe how you could redesign the listener interface for `RangeWithBoundChangeListeners` to make such a listener easy to implement. Be specific about what definitions you would change.

Name: _____

6. (20 points) This problem makes ranges generic. This code gets you started:

```
interface RangeElementTaker<T> {
    void m(T i);
}
interface RangeElement<T> extends Comparable<T> {
    T next();
}
class GenericRange<T extends RangeElement<T>> {
    private T low;
    private T high;
    public GenericRange(T lo, T hi) {
        low = lo;
        high = hi;
    }
    ...
}
```

- (a) Add implementations of `getLow`, `getHigh`, `setLow`, `setHigh`, and `forEach` to `GenericRange`. Do *not* implement `toArray`.

- (b) While it is possible to implement `toArray` correctly in `GenericRange`, it is somewhat more difficult than `forEach` for a couple specific reasons. Describe one such reason.

Part (c) is on the next page.

Name: _____

- (c) Define a class `RangeInteger` such that `new GenericRange<RangeInteger>(new RangeInteger(3), new RangeInteger(10))` typechecks and behaves like `new Range(3, 10)`. Hint: Your class will need an `int` field, which for simplicity you can make `public` rather than defining a getter method.

Name: _____

7. (12 points) The `InternedRange` subclass of `Range` defined here is a bad idea. It is an attempt at using the interning design pattern to make sure that no two instances of `InternedRange` have identical bounds. But it has multiple defects.

```
class InternedRange extends Range {
    private static HashMap<InternedRange,InternedRange> internedRanges =
        new HashMap<InternedRange,InternedRange>();

    public static InternedRange rangeFactory(int lo, int hi) {
        InternedRange r = new InternedRange(lo,hi);
        if(internedRanges.containsKey(r)) {
            return internedRanges.get(r);
        }
        internedRanges.put(r,r);
        return r;
    }

    public InternedRange(int lo, int hi) {
        super(lo,hi);
    }
}
```

- (a) The code above does not require clients to use the factory method to get an `InternedRange`. How would you change the implementation to require this?
- (b) Explain in roughly 1 English sentence why the call to `containsKey` in the code always returns `false`.
- (c) Explain in roughly 1–2 English sentences the undesirable behavior that results from the call to `containsKey` in the code always returning `false`.
- (d) Even if you fixed the problem in parts (b) and (c) (which we are *not* asking you to do), explain in roughly 1–2 English sentences why the interning design pattern, even if implemented correctly, is inappropriate for the `Range` ADT as defined in this exam.

Name: _____

8. (15 points) We consider this generic static method using the generic `List` definition in Java's standard library:

```
static <T1> void foo(List<T1> x, List<T1> y, T1 z, boolean b) {
    if(b) {
        z = y.get(0);
    }
    x.set(0,z);
}
```

- (a) For each of the following potential changes to the first line of `foo`, choose one of the following:
- A. `foo` no longer type-checks
 - B. `foo` still type-checks, all calls to `foo` that used to type-check still do, and no new calls to `foo` type-check
 - C. `foo` still type-checks, all calls to `foo` that used to type-check still do, and some additional calls to `foo` that did not used to type-check now do
 - D. `foo` still type-checks, but some calls to `foo` that used to type-check no longer do
- i. `static <T1, T2 extends T1> void foo(List<T2> x, List<T1> y, T1 z, boolean b){`
 - ii. `static <T1, T2 extends T1> void foo(List<T1> x, List<T2> y, T1 z, boolean b){`
 - iii. `static <T1, T2 extends T1> void foo(List<T1> x, List<T1> y, T2 z, boolean b){`
 - iv. `static <T1, T2 extends T1> void foo(List<T2> x, List<T2> y, T1 z, boolean b){`
 - v. `static <T1, T2 extends T1> void foo(List<T2> x, List<T1> y, T2 z, boolean b){`
 - vi. `static <T1, T2 extends T1> void foo(List<T1> x, List<T2> y, T2 z, boolean b){`
 - vii. `static <T1, T2 extends T1> void foo(List<T2> x, List<T2> y, T2 z, boolean b){`
- (b) Which of the potential changes in part (a) has the same meaning as this potential change?
- ```
static <T1> void foo(List<T1> x, List<? extends T1> y, T1 z, boolean b){
```
- (c) Complete this change to the first line of `foo` such that `foo` still type-checks and supports more client calls than any of the other variations considered thus far. (Do not write/change the method body, only the first line of the method.)
- ```
static <T1, T2 extends T1, T3 extends T2> void foo(...){
```
- (d) Give one final first-line for `foo` that uses a wildcard and allows the same client calls as your answer to part (c).

Name: _____

9. (26 points) (Short Answer, continues on to next page)

- (a) For each of the following techniques, answer “yes” if it designed to reduce the distance from defect to failure and “no” otherwise.
- Checking the representation invariant at the beginning of every public method in an ADT
 - Improving documentation as you inspect code during debugging
 - Automatically running test suites every time code is committed to version control
 - Keeping a log of experiments you try during debugging
 - Adding assertions to your code during debugging
- (b) Which statement best describes the proper way in React for a component c to pass data to its parent p , i.e., the component that contains c ?
- c uses `setState` to modify a field of an object shared by c and p
 - c passes a Prop to p that p uses to update p 's state
 - p provides a Javascript function to c via a Prop and c can call the function to update p 's state
 - c calls p 's `componentDidUpdate`
- (c) Which statement best describes how the client and the server in your campusPaths application communicate?
- The client encodes the source and destination as a URL and the server encodes the path as a JSON object.
 - The server encodes the source and destination as a URL and the client encodes the path as a JSON object.
 - The client encodes the source and destination as a JSON object and the server encodes the path as a URL.
 - The server encodes the source and destination as a JSON object and the client encodes the path as a URL.
- (d) For each of the following, answer “yes” if the caller can assume the operation has completed as soon as the caller continues executing and “no” if the caller cannot assume that.
- In React, calling `setState` to update part of a component's state
 - In React, fetching data from a web server
 - In Java, using `new` to create an instance of an anonymous inner class
 - In (non-React) JavaScript, passing an array to a function so that the function can return multiple values by adding them to the array via assignment statements

Name: _____

- (e) Which statement about the Model/View/Controller design pattern is *not* accurate?
- i. It *decouples* the model, the view, and the controller.
 - ii. An implementation usually also uses the Observer design pattern.
 - iii. The view needs to be built on top of a graphical user interface (GUI) library.
 - iv. The view and the controller may or may not be implemented in the same programming language.
- (f) For each of the following, answer “yes” if the design pattern is intended to reduce the amount of memory used by an application and “no” otherwise
- i. Interning
 - ii. Builder
 - iii. Flyweight
 - iv. Adapter
 - v. Visitor
- (g) Which of the following is *not* a benefit of structuring a large application in terms of a software architecture?
- i. Communicating to other developers the code structure using agreed-upon terminology
 - ii. Forbidding forms of communication among modules that would increase coupling
 - iii. Easier regression testing
 - iv. Better modularity
- (h) In at most one English sentence, what is the simplest way to estimate the *cost* of developing a software system?
- (i) Yes or no (no explanation): Is it a good idea in practice to mix aspects of top-down implementation and bottom-up implementation?

Name: _____

This page is blank. If you need the space here to complete your answers to a problem, please do so, but indicate on the page with the problem that the graders need to look here.

Rip this page out and do not turn it in.

When the exam refers to “the Range class,” it means the class definition in this code:

```
interface IntTaker {
    void m(int i);
}

class Range {
    private int low;    // lower bound of range
    private int high;  // upper bound of range

    public Range(int lo, int hi) {
        low = lo;
        high = hi;
    }
    public int getLow() {
        return low;
    }
    public int getHigh() {
        return high;
    }
    public void setLow(int lo) {
        low = lo;
    }
    public void setHigh(int hi) {
        high = hi;
    }
    public int[] toArray() {
        int[] ans = new int[high-low+1];
        for(int i=0; i <= high-low; i++) {
            ans[i] = low+i;
        }
        return ans;
    }
    public void forEach(IntTaker it) {
        for(int i=low; i <= high; i++) {
            it.m(i);
        }
    }
}
```