Prelim 1

CS 2110, 1 April 2021

	1	2	3	4	5	6	Total
Question	Name	OO	Loop invariants	Complexity	Recursion	Short answer	
Max	1	25	15	9	12	38	100

- This exam is open-note. Collaboration is strictly forbidden. Do not share any portion of your exam with anyone. Do not discuss the exam with anyone besides the instructors until after grades have been released.
- This exam is designed to take no more than **90 minutes**. If you have SDS accommodations, allocate additional time proportional to this duration (e.g. those granted 50% extra time should plan to spend 135 minutes on the exam). Do not spend more than this nominal time working on the exam problems. It is *your responsibility* to manage your time so that you can print, work for the nominal time, scan, and upload by the deadline while accommodating conflicts and technical hiccups.
- Submit to Gradescope *before* the end of the submission window. There will be a short grace period with late penalty to accommodate stragglers. Aim to submit early, and contact the instructors immediately if you encounter major technical difficulties.
- **Write your answers in the boxes or spaces provided**. Ambiguous answers will be considered incorrect. You should be able to fit your answers easily into the allocated space.
- In some places we have abbreviated or condensed code to reduce the number of pages that must be printed for the exam. In others, code has been obfuscated to test your understanding of the language. This does not mean that it's good style.
- Policy details and submission instructions are available on Canvas. Good luck!

Academic Integrity Statement: I pledge that I have neither given nor received any unauthorized aid on this exam. I will not talk about the exam with anyone in this course who has not yet taken the prelim.

_____

(signature)

1. Name & signature (1 point)

Is your name at the top of this page and your NetID at the top of pages 1..8 correct? If either is incorrect, or if you are missing pages, do not proceed; contact the instructors immediately! Sign the Academic Integrity Statement before submitting your exam.

2. Object-Oriented Programming (25 points)

(a) 3 points. Abstract class `MutInt`, to the right, represents a mutable wrapper class for type `int`. It has another feature: It saves the initial value of the `int` when a new-expression is evaluated; look at method `diff` to the right. The abstract class does not provide a field for the current value; subclasses must do that.

Since `MutInt`'s methods are all public and abstract, why can't `MutInt` be an interface instead of an abstract class? Put your answer in the box below.

Because it requires an instance field, in,
and interfaces cannot not have such a field.
It also has a constructor (to initialize the field),
and interfaces can't have a constructor.

```
abstract class MutInt {
    /** Initial value of the int */
    protected final int in;

    /** Constr: An instance for i */
    public MutInt(int i) { in= i; }

    /** Return this integer */
    public abstract int toInt();

    /** Add k to this int */
    public abstract void add(int k);

    /** = the current value minus
      *   the initial value */
    public abstract int diff();
}
```

(b) 12 points. Below is subclass MutInt1 of abstract class MutInt. Complete the bodies of the three methods in the boxes shown. Look at the abstract class for the specifications of the methods.

```
class MutInt1 extends MutInt {
  private int k; // The integer

  /** Constr: An instance for k */
  public MutInt1(int k) {
      super(k);
      this.k= k;
  }

  public void add(int k) {
      this.k= this.k + k;
  }

  public int toInt() { return k; }

  public int diff() {
      return k - in;
  }
}
```

(c) 10 points. In some situations, the input and output of a calculation will be an `int` but the calculation itself may require values of type `long`. Below is a second subclass of abstract class `MutInt` that keeps the value as a `long`, but when the value is requested, it is changed to an `int` and returned. Complete the bodies of methods as indicated. In the last two, use method `notInt`.

```
public class MutInt2 extends MutInt {
  private long k; // The integer

  public MutInt2(int k) { ... } // Constructor

  public void add(int k) {... }

  /** Return true if k is not in the range of type int */
  private static boolean notInt(long k) {
```

```
    return k < Integer.MIN_VALUE || Integer.MAX_VALUE < k;
```

```
  }

  /** Return this integer. Throw RuntimeException if it is not an int. */
  public int toInt() {
```

```
    if (notInt(k)) throw new RuntimeException();
    return (int) k;
```

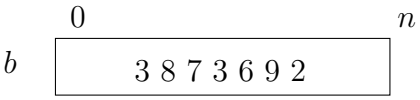
```
  }

   /** Return (current value) - (initial value), but throw a
     * RuntimeException if the value to be returned is not an int. */
  public int diff() {
```

```
    if (notInt(k - in)) throw new RuntimeException();
    return (int) (k - in);
```

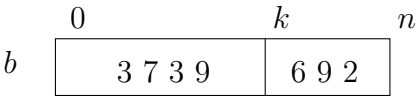
```
  }
}
```

3. Loop Invariants (15 points).

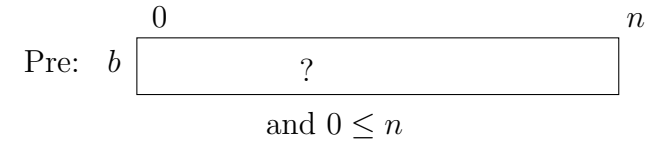
We want a loop, with initialization, that copies the odd values in array segment $b[0..n-1]$ to the beginning of the array segment while preserving their order relative to each other. Consider this array segment:

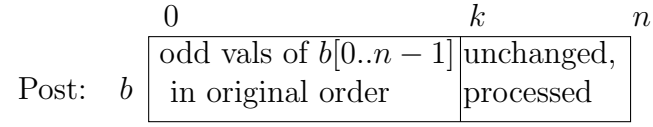


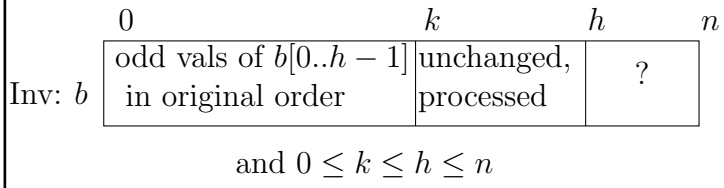
Below is the result. The k odd values have been copied to the beginning while remaining in the same order as before they were processed, and $b[k..n-1]$ is unchanged.



To the right are the precondition, postcondition, and invariant for this problem.







(a) 3 points In the box to the right, write the initialization for the loop. Assume that variables have already been declared and that input parameters have been provided. Be sure to look at the constraints on h and k mentioned in the invariant.

```
h= 0;
k= 0;
```

(b) 3 points The loop has the form **while** (B) *repetend*. To the right, write condition B. Do *not* write anything else.

```
h != n  OR  h < n
```

(c) 3 points In the box to the right, write a single statement for the repetend that makes progress toward termination. Do *not* write a while loop; just the single statement.

```
h= h + 1;
```

(d) 6 points In the boxes to the right, complete the if-statement to be put in the repetend *before* `part c` to ensure that the invariant is true after the statement in part c is executed.

```
if ( b[h] % 2 != 0 ) {
    b[k]= b[h];
    k= k+1;
}
```

`part c`

4. Complexity 9 Points.

(a) 3 points. Consider method `fee`, below. In the box to the right, give the tightest expression, in big-O notation, for how the time complexity of a call `fee(n)` scales with `n`.

```
public int fee(int n) {
   int ctr= 0;

   for (int i= 0; i < 10; i++)
      for (int j= i + 1; j < 10; j++)
         for (int k= 2; k < n; k++)
             ctr= ctr + i + j + k;

   return ctr;
}
```

$O(n)$

(b) 6 points. For each function $f(n)$ below, in the box to the right, give the smallest set $O(g(n))$ that contains $f(n)$.

1. $f(n) = n^2/2 + n \log n$

$O(n^2)$

2. $f(n) = 2^{n-5} + 7n^3$

$O(2^n)$

3. $f(n) = 5n + 500/n$

$O(n)$

5. Recursion (12 Points)

Complete function `num` below. See the examples to the right. In the last example, the answer is 3 because the integer (645222) ends with 3 2's.

Do not use a loop. Do not use Strings. The int operators / and % may be useful. If you are contemplating declaring a local variable, don't. Just tackle the base case(s) and then the recursive case(s).

num(6, 3) = 0
num(753, 2) = 0

num(6, 6) = 1
num(33353, 3) = 1

num(645222, 2) = 3

```
/** = the number of times d appears in a row at the least-
  *   significant end of the decimal representation of n.
  * Precondition: 0 <= n   and   0 <= d < 10. */
 public static int num(int n, int d) {
```

```
if (n % 10 != d) return 0;
if (n < 10) return 1;
return 1 + num(n / 10, d);
```

}

6. Short Answer (38 points)

(a) 8 points. Classes S and D appear to the right. Below, for each pair of methods, identify whether they are an example of (A) overloading, (B) overriding, or (C) neither by writing *A*, *B*, or *C* in the box to the right.

```
class S { ... }

class D extends S { ... }
```

public int doStuff(int x) {...} in class S
public int doStuff(int x) {...} in class D

B

public double readStuff(S s) {...} in class S
public double readStuff(D d) {...} in class D

A

public int sayStuff() {...} in class D
public int sayStuff(String s) {...} in class D

A

public boolean fooStuff() {...} in class S
public int fooStuff() {...} in class S

C

(b) 9 points. Exception handling. The two methods to the right have four print statements that print parts of the word *Cornell*:

''rn'', ''ell'', ''Co'', ''ell''.

Note: method `first` calls method `second`.

In the box below, write a sequence of three calls on method `first` that, together, print *Cornell*. Do *not* put in any calls on method `second`.

first(2, 2); // arg 1, >= 0. arg, 2.
first(-1, 0); // arg 1, < 0. arg 2, anything.
first(1, 1); // arg 1, >= 0. arg 2, 1.

```
static void first(int i1, int i2) {
  try {
    if (i1 < 0) {
      System.out.print("rn");
    } else {
      throw new ArithmeticException();
    }
  } catch (RuntimeException e) {
    second(i2);
  } catch (Throwable e) {
    System.out.print("ell");
} }

static void second(int i3) {
  try {
    assert i3 > 0;
    if (i3 == 2 || i3 / (i3-1) == 0) {
      System.out.print("Co");
    }
  } catch (ArithmeticException e) {
    System.out.print("ell");
} }
```

(c) 9 points. Class *Wrap* is to the right. Specs have been removed to make it easier to see the code. Below is a code snippet. Place your answers to the questions following it in the boxes provided.

```
L1: Object w= new Wrap<Object>();
L2: ((Wrap) w).set(1, "Student");
```

```
public class Wrap<E> {
  private E val;
  private String author;

  public void set(E v, String n)
      { val= v; author= n; }

  public E get() { return val; }
}
```

What is the static type of `w` ?

Object

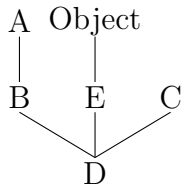
What is the dynamic type of `w` just after statement L1 has been executed?

Wrap

What is the dynamic type of the value that would be returned by `((Wrap)w).get()` immediately after statement L2 has been executed?

Integer

(d) 12 points. The diagram to the right depicts an object of class `D`. Answer the questions below, placing your answers in the boxes. In writing a declaration of a class or interface, use {} for the body and do *not* write an access modifier (e.g. public). For example, for E, we would write simply `class E{}` .



1. Write a declaration for `D`:

class D extends E implements B, C {}

2. Write a declaration for `C`:

interface C {}

3. Write a declaration for `B`:

interface B extends A {} Note: implements is wrong

4. Write a declaration for `A`:

interface A {}

5. Suppose `A` contains a declaration `public void m();` and `E` contains method `m` shown below. Will this compile? (yes or no)

```
public void m() { A d= (A) (new E()); }
```

yes

6. Suppose `A` contains a declaration `public void m();` and `E` contains method `m` shown below. Will this compile? (yes or no)

```
public void m() { A d= new E(); }
```

no