Name:_____ UNM Username:_____

Answer all questions in the space provided. Write clearly and legibly, you will not get credit for illegible or incomprehensible answers. Print your name at the top of every page.
This is a closed book exam. However, each student is allowed to bring one page of notes to the exam. Also, you are permitted the use of a "dumb" calculator to perform basic arithmetic.

Question:	1	2	3	4	5	6	7	8	Total
Points:	10	7	14	15	18	14	12	10	100
Score:									

1. Consider the following classes.

   For each specified method, does the method in `Child` override, overload, or hide the method in `Parent`?

```
public class Parent {
  public static void methodA(int i) {}
  public void methodB(int i) {}
  protected void methodC(int a, String b) {}
  protected void methodD(String s) {}
  protected static void methodE(int i) {}
}
```

```
public class Child extends Parent {
  public static void methodA(int i) {}
  public void methodB(int x) {}
  public void methodC(String a, int b) {}
  public void methodD(String x) {}
  public static void methodE(String s) {}
}
```

   (a) `methodA` ◯ override ◯ overload ◯ hide (2)

   (b) `methodB` ◯ override ◯ overload ◯ hide (2)

   (c) `methodC` ◯ override ◯ overload ◯ hide (2)

   (d) `methodD` ◯ override ◯ overload ◯ hide (2)

   (e) `methodE` ◯ override ◯ overload ◯ hide (2)

2. Indicate if the following statements are true or false.

   (a) ◯ True ◯ False Every object has a `toString` method. (1)

   (b) ◯ True ◯ False Every exception *must* be caught with a try/catch statement. (1)

   (c) ◯ True ◯ False An abstract class may contain member variables. (1)

   (d) ◯ True ◯ False In the expression `3 < 4 && foo(5)`, method `foo` will be called with an argument of 5. (1)

   (e) ◯ True ◯ False It is possible to return an `Exception` from a method. (1)

   (f) ◯ True ◯ False A `NullPointerException` is a checked exception. (1)

   (g) ◯ True ◯ False The `HashMap` class implements the `Collection` interface. (1)

3. Select the *single best* answer for each of the following questions.

(a) What is the value of the following expression? (2)

```
1 + 2 + "3"+ 4 * 5 + 6
```

A. `123456` E. `"123206"` I. Some other value.
B. `21` F. `"123456"` J. The value of this expression is undefined.
C. `42` G. `"33206"` K. This expression would result in a compilation error.
D. `56` H. `"3326"`

(b) Which interface would be the best choice to associate course numbers with the names of the instructors teaching them? (2)

A. `Collection` B. `Deque` C. `List` D. `Map` E. `Set` F. `Queue`

(c) Which type could `foo` be in the following code snippet? (2)

```
int n = foo.size();
```

A. `Collection` B. `Deque` C. `List` D. `Set` E. `Queue` F. `SortedSet`
G. Any of these. H. None of these.

(d) Which type could `foo` be in the following code snippet? (2)

```
Object x = foo.get(0);
```

A. `Collection` B. `Deque` C. `List` D. `Set` E. `Queue` F. `SortedSet`
G. Any of these. H. None of these.

(e) Which of the following types could `foo` *not* be in the following code snippet? (2)

```
foo.add("hello");
```

A. `Collection` B. `Deque` C. `List` D. `Map` E. `Queue` F. `Set`
G. Any of these. H. None of these.

(f) What is printed when the following code is compiled and executed? (2)

```
public class StringCompare {
    public static void main(String[] args) {
        String s1 = new String("Test");
        String s2 = new String("Test");
        if (s1==s2) System.out.print("Same ");
        if (s1.equals(s2)) System.out.print("Equals");
    }
}
```

A. `Equals` D. The code compiles, but nothing is displayed upon execution.
B. `Same` E. Some other output.
C. `Same Equals` F. The output of this program is undefined.
G. The code fails to compile.

(g) Code that is generalized to work for various data types specified by a type parameter is known as what? (2)

A. abstract B. final C. generic D. polymorphic E. static

4. Do the following code snippets compile and run without error? Answer yes or no. If you answered no, explain what is wrong.

(a) (3)

```
Collection<double> numbers = new TreeSet<>();
```

(b) (3)

```
Map<String, String> map = new HashMap<>()
```

(c) (3)

```
Double default = 42;
```

(d) (3)

```
public class MyClass {
  private static int x = 1.5;

  public static void main(String[] args) {
    x++;
    System.out.println(x);
  }
}
```

(e) (3)

```
List<String> names = new ArrayList<String>();
names.put("Jane");
```

5. Consider the following interface.

```
public interface CounterInterface {
  void increment();
  int getValue();
}
```

For each of the following classes:

- Does it correctly implement the interface?
- If it does not, what is wrong with it?

(a) (3)

```
public class Counter implements CounterInterface {
  public void increment() {}
  public int getValue() { return 42; }
}
```

(b) (3)

```
public class Counter implements CounterInterface {
  int value = 0;

  void increment() { value++; }
  int getValue() { return value; }
}
```

(c) (3)

```
public class Counter extends CounterInterface {
  private int value;

  public Counter(int n) {
    value = n;
  }

  public void increment() { value++; }
  public int getValue() { return value; }
}
```

(d) (3)

```
public class Counter implements CounterInterface {
  private int value = 0;

  public void increment(int n) { value += n; }
  public int getValue() { return value; }
}
```

(e) (3)

```
public class Counter implements CounterInterface {
  private final int value = 0;

  public void increment() { value++; }
  public int getValue() { return value; }
}
```

(f) (3)

```
public class Counter implements CounterInterface {
  private int i, n;

  public Counter(int i, int n) {
    this.i = i;
    this.n = n;
  }

  public void increment() { i += n; }
  public int getValue() { return i; }
}
```

6. Consider the following classes. What is the output of this code? (14)

```
public class Foo {
    protected double x;
    protected String y;

    public Foo() {
        this("midterm");
    }

    public Foo(String x) {
        this(x, x.length());
    }

    public Foo(String x, int y) {
        this.x = y * 0.5;
        this.y = x;
    }

    public void stuff() {
        System.out.println(x);
        System.out.println(y);
    }

    public void stuff(int x) {
        stuff("CS" + x);
    }

    public void stuff(String y) {
        x -= 2;
        System.out.println(x);
        System.out.println(y);
    }
}
```

```
public class Bar extends Foo {
    protected int z = 2023;

    public Bar(String z) {
        System.out.println(x);
        System.out.println(y);
        System.out.println(z);
    }

    public void stuff() {
        super.stuff("exam");
        System.out.println(z);
    }

    public void stuff(String x) {
        stuff();
        System.out.println(x);
    }

    public static void main(String[] args) {
        Foo obj = new Bar("time");
        obj.stuff(251);
    }
}
```

7. Consider the following class. What is the output of this code? (12)

```
public class Baz {
    private static String a = "boo";
    private int b = -100;

    public Baz(String b) {
        a = b;
        this.b = a.length();
    }

    public void doThings(String c) {
        System.out.println(a);
        System.out.println(b);
        a = c;
        b += a.length();
    }

    public static void main(String[] args) {
        Baz bob = new Baz("banana");
        bob.doThings("strawberry");

        Baz jane = new Baz("kiwi");
        jane.doThings("pear");

        bob.doThings("peach");
        jane.doThings("pineapple");

        System.out.println(a);
        System.out.println(bob.b);
        System.out.println(jane.b);
    }
}
```

8. Write a method that takes a `Collection` of `String` objects (this should take any type of collection, not just a specific implementation) and returns the length of the shortest `String`. In other words, return the minimum of the lengths of the `String`s in the `Collection`. If the collection is empty, return `Integer.MAX_VALUE`. (This is a constant in the `Integer` class that is the largest value that can be stored in an integer. You may assume that no `String` in the input will be longer than this.) (10)

   To do this, select only the necessary lines of code from the choices below, copy them in the correct order, and add indentation and closing curly braces as needed to make a correct Java method with the desired behavior.

   A. `public static int minLength(Collection<String> strs) {`
   B. `public static int minLength(List<String> strs) {`
   C. `for(int i = 0; i < strs.size(); i++) {`
   D. `for(String s : strs) {`
   E. `if (min < len) {`
   F. `if (len < min) {`
   G. `min = len;`
   H. `len = min;`
   I. `return min;`
   J. `return len;`
   K. `int len = strs.size();`
   L. `int len = s.length();`
   M. `len++;`
   N. `int min = 0;`
   O. `int min = Integer.MAX_VALUE;`
   P. `String s = strs.get(i);`
   Q. `String s = strs[i];`