# CS 1331 Exam 2 Practice

# ANSWER KEY

- Signing signifies you are aware of and in accordance with the **Academic Honor Code of Georgia Tech**.
- Calculators and cell phones are NOT allowed.
- This is an object-oriented programming test. Java is the required language. Java is case-sensitive. DO NOT WRITE IN ALL CAPS. A Java program in all caps will not compile. Good variable names and style are required. Comments are not required.

| Question | Points per Page | Points Lost | Points Earned | Graded By |
|---|---|---|---|---|
| Page 1 | 0 | - | = | |
| Page 2 | 0 | - | = | |
| Page 3 | 0 | - | = | |
| Page 4 | 0 | - | = | |
| Page 5 | 0 | - | = | |
| Page 6 | 0 | - | = | |
| Page 7 | 0 | - | = | |
| TOTAL | **??** | - | = | |

1. **True or False**

In each of the blanks below, write "T" if the statement beside the blank is true, "F" otherwise.

[2] (a) __T__ In Java, every class you write is a subclass of at least one other class.

[2] (b) __T__ In a constructor, if an explicit `super` call is present, it must be the first statement in the constructor.

[2] (c) __T__ If a class defines a single constructor, the constructor contains an implicit `super` call if no explicit `super` call is provided in the constructor.

[2] (d) __T__ You can define a subclass of an abstract class without defining any of the abstract methods defined in the superclass.

[2] (e) __T__ In a concrete class that implements an interface, you must provide definitions for all of the methods declared in the interface.

[2] (f) __T__ Overloading a superclass method in a subclass means defining a method with the same name as the superclass method but with a different parameter list.

[2] (g) __T__ `protected` members are visible to classes in the same package and to subclasses.

[2] (h) __T__ `private` members are visible in the class in which they are defined, but not in subclasses.

[2] (i) __T__ `FileNotFoundException` is a checked exception.

[2] (j) __T__ In a try statement with multiple `catch` clauses, the first `catch` clause that can catch the exception thrown in the corresponding `try` block will be exectued.

2. **Multiple Choice** Circle the letter of the correct choice.

[2] (a) In which package is `Object` from the standard library located?

A. java.util
**B. java.lang**
C. java.text
D. java.object

[2] (b) In a class named `Pill`, what is the correct declaration for a method that overrides the `equals` method defined in `Object`?

A. public boolean equals(Pill other)
**B. public boolean equals(Object other)**
C. protected boolean equals(Pill other)
D. protected static boolean equals(Object other)

[2] (c) A method declared in a superclass is said to be polymorphic in its subclasses if _____.

A. the method is declared `final` in the superclass
**B. the method is overriden in the subclasses**
C. the method is overloaded in the subclasses
D. the method chains to the superclasses using `super`

[2] (d) Which of the following features is required for a language to be called an object-oriented langauge?

A. separate compilation
**B. dynamic method binding**
C. lazy evaluation
D. higher-order functions

[2] (e) How many classes may a class extend?

A. 0
**B. 1**
C. 2
D. $[0, \infty)$

3. **Multiple Choice** Circle the letter of the correct choice.

Given the following class definitions:

```
public abstract class Animal {
  public abstract void speak();
}
```

```
public class Mammal extends Animal {
  public void speak() {
    System.out.println("Hello!");
  }
}
```

```
public class Dog extends Mammal {
  public void speak() {
    System.out.println("Woof, woof!");
  }
}
```

```
public class Cat extends Mammal {
  public void speak() {
    System.out.println("Meow!");
  }
}
```

[2] (a) Which of the following statements will **not** compile?

A. Animal mittens = new Cat();

**B. Animal house = new Animal();**

C. Animal farm = new Mammal();

[2] (b) Which of the following statements will **not** compile?

A. Mammal fido = new Dog();

**B. Dog fido2 = fido;**

C. ((Mammal) fido).speak();

[2] (c) Assuming the statement `Mammal fido = new Dog();` has been executed, what does `fido.speak()` print?

A. Hello!

**B. Woof! Woof!**

C. Meow!

[2] (d) Assuming the statement `Mammal fido = new Dog();` has been executed, what does `((Mammal) fido).speak()` print?

A. Hello!

**B. Woof! Woof!**

C. Meow!

[2] (e) Assuming the statement `Mammal sparky = new Mammal();` has been executed, which of the following statements will compile but cause a `ClassCastException` at run-time?

A. Mammal fido = new Dog();

**B. Dog huh = (Dog) sparky;**

C. Dog fido2 = (Dog) new Dog();

[10] 4. **Tracing**

Consider the following code:

```
public class Wee {

    static void bar() throws Throwable {
        throw new Throwable("Wee!");
    }

    static void foo() throws Throwable {
        bar();
        System.out.println("Foo!");
    }

    public static void main(String[] args) {
        try {
            foo();
        } catch (Throwable t) {
            System.out.println(t.getMessage());
        }
        System.out.println("I'm still running.");
    }
}
```

What is printed when main is executed?

**Solution:**

```
Wee!
I'm still running.
```

[10] 5. **Tracing**

Given the following class definitions:

```
public class Super {
  protected int x = 1;

  public Super() {
    System.out.print("Super");
  }
}
```

```
public class Duper extends Super {
  protected int y = 2;

  public Duper() {
    System.out.println(" duper");
  }
}
```

```
public class Fly extends Super {
  private int z, y;

  public Fly() {
    this(0);
  }

  public Fly(int n) {
    z = x + y + n;
    System.out.println(" fly times " + z);
  }

  public static void main(String[] args) {
    Duper d = new Duper();
    int delta = 1;
    Fly f = new Fly(delta);
  }
}
```

What is printed when `Fly` is run?

```
Super duper
Super fly times 2
```

6. **Short Answer**

[4] (a) Write the header for a class named `Foo` that extends a class called `Bar` and implements two interfaces, `Baz` and `Bang`.

**Solution:** `public class Foo extends Bar implements Baz, Bang`

[4] (b) Assume you have two variables of type `Foo` and `Foo` is properly written. The variables are named `f1` and `f2`. Write the expression that represents whether or not the objects that `f1` and `f2` reference have the same value, by the `Foo` class's definition of equal value.

**Solution:** `f1.equals(f2)` or `f2.equals(f1)`

[4] (c) Assume you have two variables of type `Foo` and `Foo` is properly written. The variables are named `f1` and `f2`. Write the expression that represents whether `f1` is an alias of `f2`.

**Solution:** `f1 == f2` or `f2 == f1`

[4] (d) Given that `FileInputStream`'s constructor throws `FileNotFoundException`, which is a subclass of `Exception`, write the header for a public method named `process` that takes a `String` parameter and returns nothing, and whose body instantiates a `FileInputStream` object and does not contain a try-catch statement.

**Solution:** `public void process(String file) throws FileNotFoundException`

[4] (e) Given a method declared as:

```
private void initFromFile(File empData) throws FileNotFoundException,
                                               IOException,
                                               ParseException
```

And the following declarations for the exception classes:

```
public class FileNotFoundException extends IOException
public class IOException extends Exception
public class ParseException extends Exception
```

Write a `try-catch` statement in which you call the `initFromFile` method and catch all the possible exceptions that might be thrown from `initFromFile`. Leave your catch clauses empty.

**Solution:**

```
try {
    initFromFile(new File(employeeDataFile)); // 1 pt. Call inside try
} catch (FileNotFoundException e) {        // 1 pt.Must be before IOException
    // ...
} catch (IOException e) {                  // 1 pt.
    // ...
} catch (ParseException e) {               // 1 pt.  Can appear any order
    // ...
}
// No points off if they include a catch clause for Exception
```

[20] 7. Given the following class and interface definitions:

```
public abstract class Pfunker implements Comparable {
    /**
     * LOLLYPOP < ATLANTEAN < CLONE < PILL < PYRAMID < FLASHLIGHT < ATOMIC_DOG
     */
    public enum Level {LOLLYPOP, ATLANTEAN, CLONE, PILL, PYRAMID,
                       FLASHLIGHT, ATOMIC_DOG}
    private Level level;
    private String name;

    public Pfunker(String name, Level level) {
        this.name = name;
        this.level = level;
    }
}
```

```
public interface Comparable {
    /**
     * Compares this object with the specified object for order. Returns a
     * negative integer, zero, or a positive integer as this object is less
     * than, equal to, or greater than the specified object.
     */
    public int compareTo(Object o);
}
```

Write the minimum concrete class named `ConcretePfunker` which is a subclass of `Pfunker`. You compare one `Pfunker` to another by comparing their `levels`. The space provided is more than sufficient. You will not be given any scratch paper. Hints:

- You may want to use `Enum`'s `ordinal()` method, which "Returns the ordinal [int] of this enumeration constant (its position in its enum declaration, where the initial constant is assigned an ordinal of zero)."
- The body of the one non-constructor method you need to write can be done in one line.

```
public class ConcretePfunker extends Pfunker {
    public ConcretePfunker(String name, Level level) {
        super(name, level);
    }

    public int compareTo(Object other) {
        return this.level.ordinal() - ((Pfunker) other).level.ordinal();
        // or return this.level.compareTo(((PFunker) other).level);

    }
}
```