```
Which method invocation in the zoo package demonstrates the concept of
polymorphism? Explain why that invocation is polymorphism.

|____|

[[ The invocation of a.birth() in the main method of the Zoo class
demonstrates polymorphism because a single instruction causes Java to
invoke different methods from a single line of code.  Note that a.eat()
and a.move() invoke the same methods for each different kind of animal -
the Animal.eat and the Animal.move methods - but have different field
values. Thus, the invocations of a.eat and a.move are not strictly
```

↳Add **Question 7.4**

✚ Add **Question 8**

Save Assignment

Q1 True/False Questions

10 Points

In the following questions, select either "True" or "False".

Q1.1

1 Point

A child class inherits its parent's static methods and may override those methods. When you run `Child.staticMethod()`, if the child class does not have a static method called "staticMethod", then and Child extends Parent, then Java will invoke `Parent.staticMethod()` for you.

○ true

◉ false

EXPLANATION

Static methods do not participate in inheritance and cannot be overridden. If you want to run `Child.staticMethod()`, you must invoke `Child.staticMethod()`. If you want to invoke `Parent.staticMethod()`, you must invoke `Parent.staticMethod()`.

Q1.2

1 Point

The most important concept in subtyping is that:

An object of a sub-type class can be placed anywhere an object of the super-type can occur.

This rule works because an object of a sub-type class contains an object of the super-type class inside of it.

- (●) true
- ( ) false

Q1.3

1 Point

If you override a method without specifying the @Override compiler annotation, the Java compiler will issue a warning message.

- ( ) true
- (●) false

EXPLANATION

The *absence* of the @Override annotation does not cause any warning messages. You may override a method without specifying @Override. In fact, we were overriding the `toString()` method many times earlier this semester without specifying @Override.

Q1.4

1 Point

If I invoke `this.super.toString()`, I will invoke the toString() method in the child class because the dynamic type of "this" is the child class, not the parent class.

○ true

◉ false

EXPLANATION

The "super" keyword affects the dynamic type of `this`, so `this.super.toString()` will invoke the parent's toString method.

Q1.5

1 Point

Every class I create in Java is actually a sub-class.

◉ true

○ false

EXPLANATION

If my class extends something, then it is a sub-class of the class it extends. If my class does not extend something, then it is a sub-class of the Object class.

Q1.6

1 Point

If an exception gets thrown inside a try block, but none of the catch blocks specify the kind of exception that was thrown, then Java executes the rest of the try block.

○ true

◉ false

EXPLANATION

If there are no catch blocks to catch the exception, it continues to percolate up the call chain. If it is never caught, it causes the program to end. Control **never** returns after an exception is thrown.

Q1.7

1 Point

Users of an interface may declare a reference variable using that interface name and, after instantiation, invoke all of the interface methods from that reference variable.

- (●) true
- ( ) false

EXPLANATION

Before the variable is used, it must be instantiated to an object in some class which implements the interface. Therefore, it is valid to assume that there are concrete implementations of the abstract methods defined in the interface.

Q1.8

1 Point

Abstract art, like an abstract method, gives a vague idea of the actual object being represented, but does not specify all the details.

- (●) true
- ( ) false

EXPLANATION

And you thought this was a computer science class!

Q1.9

1 Point

It is standard practice to declare a field or local variable as one of the Collection infrastructure interfaces, such as List, but instantiate that variable using a concrete implementation class such as ArrayList. This makes it easy to replace the concrete implementation without editing lots of code.

- (●) true
- ( ) false

EXPLANATION

EXPLANATION

All that needs to be changed is the initial instantiation.

Q1.10

1 Point

Ripley's Believe it or Not: A Java TreeMap is neither a tree, nor is it a map. It is actually a concrete implementation of one of the Java interfaces that deals with key/value pairs, and has a backing store of a data structure characterized by nodes with left and right sub-nodes.

- (●) true
- ( ) false

EXPLANATION

Lawyers will argue with me... a TreeMap is a Map with a backing store of Tree, but it has nothing to do with plants that grow to be very large, or those things we used to use before we had GPS to find out where to go.

Q2

5 Points

Suppose the Duck class extends the Bird class, and you code the following:

```
Bird bird;
Duck quack = new Duck();
bird = quack;
```

When you assign `quack` to `bird`, you are limiting what you can do... you can no longer do "Duck" things to bird, you can only do "Bird" things. Why would you throw away capabilities in your program?

- ( ) Birds can do everything Ducks can do anyway
- ( ) How else would you demonstrate the concept of sub-typing? There's no other good reason to do this.
- (●) You want to evaluate different kinds of birds, Duck being just one subclass.
- ( ) You will explicitly downcast bird back to Duck before you use it, and get those

- You will explicitly downcast bird back to Duck before you use it, and get those capabilities back.

Q3

5 Points

Why is the following code misleading and incorrect?

```
IllegalArgumentException badArg = new IllegalArgumentException("Incorrect input value")
...

if (arg1 > 7) throw badArg;
...
if (arg2%30 < 14) throw badArg;
...
```

- It is incorrect to throw the same exception twice.
- You can never evaluate arg2%30 because after you have thrown an exception, the method stops.
- If you throw badArg, the stack trace reports the line where badarg was declared as the problem line.
- The error message for arg1 is incorrect when arg2 is wrong.
- It would be better to print a message and keep processing if the argument is incorrect.

Q4

10 Points

Which of the following statements are correct statements about the relationship between interfaces and classes?

Q4.1
2 Points

Many different classes may implement a single interface.

- [x] true
- [ ] false

Q4.2
2 Points

A single class may implement many different interfaces.

- [x] true
- [ ] false

Q4.3
2 Points

An interface type may be used anywhere a class that implements that interface appears.

- [ ] true
- [x] false

EXPLANATION

It's the other way around - a class that implements an interface may be used anywhere that interface appears. The implementing class is the sub-type; the interface is the super-type.

Q4.4
2 Points

An interface may only implement one lower level interface.

- [ ] true

true

◉ false

EXPLANATION

An interface may implement as many lower level interfaces as it wants to.

Q4.5

2 Points

An interface may be used as the static type of a reference variable, but can never be used as the dynamic type of that reference variable.

◉ true

○ false

EXPLANATION

In order to be the dynamic type, the reference variable must point to an object of the interface type - but there cannot be objects of the interface type because the interfaces is abstract.

Q5

18 Points

Given the following Java Code:

In Buggy.java:

```
1: package test02;
2: public class Buggy {
3:      private int wheels;
4:      public Buggy() { this.wheels = 4.0; }
5:      @Override public String toString() {
6:              return "Buggy with " + wheels + " wheels.";
7:      }
8: }
```

In Wagon.java:

```
 1: package test02;
 2: public class Wagon extends Buggy {
 3:     private int cap;
 4:     public Wagon() { cap=500; super(); }
 5:     @Override public String toString() {
 6:             return "Wagon with " + wheels + " wheels can carry " + cap ;
 7:     }
 8:     public static void main(String[] args) {
 9:             System.out.println("Made " + new Wagon());
10:     }
11:}
```

Which of the following describes bugs in this code?

Q5.1

3 Points

Buggy.java:4 this.wheels undefined

- ○ Is a bug
- ◉ Is not a bug

EXPLANATION

`wheels` is defined in Buggy.java:3 as a field, and can be referenced using `this.wheels`.

Q5.2

3 Points

Buggy.java:4 Narrowing conversion, cannot convert from double to int

- ◉ Is a bug
- ○ Is not a bug

EXPLANATION

Since `double` is wider than `int`, the compiler will issue an error message if you try to

assign the `double` value 4.0 to the `wheels` field which is declared as `int`. Should be `wheels=4;`.

Q5.3

3 Points

Wagon.java:4 cap is undefined

- ○ Is a bug
- ◉ Is not a bug

EXPLANATION

`cap` is defined as a field on Wagon.java:3, and since there is no local variable or parameter named `cap`, Java assumes the reference to `cap` in Wagon.java:4 is a reference to `this.cap`.

Q5.4

3 Points

Wagon.java:4 Constructor call must be the first statement in a constructor

- ◉ Is a bug
- ○ Is not a bug

EXPLANATION

If the child constructor implicitly invokes the parent constructor, that **must** come first in the child constructor.

Q5.5

3 Points

Wagon.java:6 The field Buggy.wheels is not visible

- ◉ Is a bug
- ○ Is not a bug

EXPLANATION

Java assumes reference to `wheels` in Wagon.java:6 is a field, `this.wheels`. Since there is no `wheels` field in the Wagon class, Java checks the parent class to find a `wheels` field. But, since we are not in the Buggy class, and the `wheels` field is private in the Buggy class, the Wagon class cannot see the `wheels` field.

Q5.6

3 Points

Wagon.java:9 new Wagon() never referenced

- ○ Is a bug
- ◉ Is not a bug

EXPLANATION

In fact, `new Wagon()` produces a reference to an object, and that reference is used as an argument to the String concatenation `+` operator.

Q6

27 Points

Given the following Java Code:

In Subconscious.java:

```
package test2;
public class Subconscious {
        private String name;
        public Subconscious(String name) { this.name = name; }
        public String getName() { return name; }
        public void promise() {
                System.out.println(name + "promises not to keep any promises.");
        }
        public void claim() {
                System.out.println(name + "will rule the world!");
        }
}
```

In Red.java:

```
package test2;
public class Red extends Subconscious {
        public Red(String name) { super(name); }
        @Override public void promise() {
                System.out.println(super.getName()+" will build a wall.");
        }
        @Override public void claim() {
                System.out.println(super.getName()+" will make America great again, aga
        }
}
```

In Blue.java:

```
public class Blue extends Subconscious {
        public Blue(String name) { super(name); }
        @Override public void promise() {
                System.out.println(super.getName()+
                        " will be a President for all Americans.");
        }
        @Override       public void claim() {
                System.out.println(super.getName()+" says it's time to battle for the s
        }
}
```

and in Debate.java:

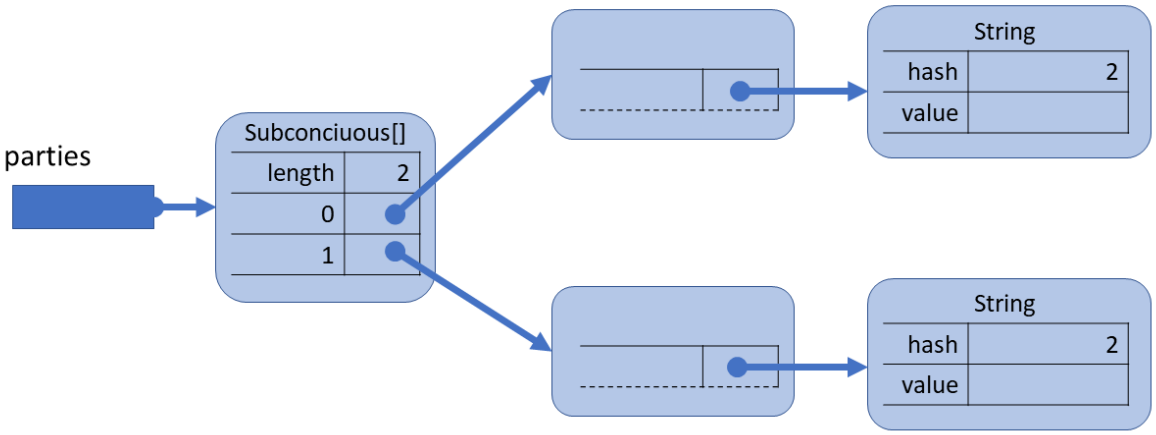
```
package test2;
public class Debate {
        public static void main(String[] args) {
                Blue dem = new Blue("Joe");
                Red rep = new Red("Donald");

                Subconscious[] parties = { rep,dem };
                for(Subconscious party : parties) {
                        party.promise(); }
                for(Subconscious party : parties) {
                        party.claim();  }
}
```

Q6.1

12 Points

If the code has run until after the parties array has been instantiate and initialized, given the following diagram:



What will be the class name in the upper middle block?

Red

What will be the field name in the upper middle block?

name

What will be the value in the upper String block?

Donald

What will be the class name in the lower middle block?

Blue

What will be the field name in the lower middle block?

name

What will be the value in the lower String block?

What will be the value in the lower string block?

Joe

Q6.2

10 Points

If you run `java test2.Debate`, what will get printed to the screen?

EXPLANATION

Donald will build a wall.
Joe will be a President for all Americans.
Donald will make America great again, again.
Joe says it's time to battle for the soul of America.

Q6.3

5 Points

The Subconscious method claim states that some politician will rule the world. But when the politicians get in a debate, none of the politicians actually claim they will rule the world. What is keeping the subconscious claim unprinted?

EXPLANATION

Both the Red and the Blue class override the subconscious claim, and since methods are chosen based on the dynamic type rather than the static type, even though the main method invokes the claim method using a Subconscious static type, the dynamic type is really Red or Blue, so we get the overridden methods.

Q7

25 Points

Given the following classes:

In Animal.java:

```
package test2;
public class Animal {
        String food; String motion;     String kids;
        public Animal(String food, String motion, String kids) {
                this.food = food;
                this.motion = motion;
                this.kids = kids;
        }
        String eat() { return "I eat " + food + "."; }
        String move() { return "I move " + motion + "."; }
        String birth() { return "I just had " + kids + "." ; }
}
```

In Tiger.java:

```
package test2;
public class Tiger extends Animal{
        public Tiger() {
                super("small animals","silently and sneakily","two cubs"); }
}
```

In Elephant.java:

```
package test2;
public class Elephant extends Animal {
        public Elephant() {
                super("leaves and grass", "with a lumbering walk","one calf"); }
}
```

In Snake.java:

```
package test2;
public class Snake extends Animal {
        public Snake() { super("mice", "on my belly", "20 hatchlings"); }
        @Override String birth() { return "I left 20 eggs to hatch."; }
}
```

In Zoo.java:

```
package test2;
public class Zoo {
	ArrayList<Animal> animals;
	public Zoo() {
		this.animals = new ArrayList<Animal>();
	}
	public boolean add(Animal e) {
		return animals.add(e);
	}
	public static void main(String[] args) {
		Zoo zoo=new Zoo();
		zoo.add(new Elephant()); zoo.add(new Tiger());  zoo.add(new Snake());
		for(Animal a: zoo.animals) {
			System.out.println(a.eat() + " " + a.move() + " " + a.birth());
		}
	}
}
```

Q7.1

10 Points

What would get printed to the screen if you run the main method from the Zoo class?

EXPLANATION

I eat leaves and grass. I move with a lumbering walk. I just had one calf.
I eat small animals. I move silently and sneakily. I just had two cubs.
I eat mice. I move on my belly. I left 20 eggs to hatch.

Q7.2

10 Points

Write a new sub-class of animal to put in a zoo. Your class should have a no-argument constructor.

EXPLANATION

```
public class Monkey extends Animal {
    public Monkey() {
        super("bananas", "by swinging on vines", "a cute baby monkey");
    }
}
```

Q7.3

5 Points

Which method invocation in the zoo package demonstrates the concept of polymorphism? Explain why that invocation is polymorphism.

EXPLANATION

The invocation of a.birth() in the main method of the Zoo class demonstrates polymorphism because a single instruction causes Java to invoke different methods from a single line of code. Note that a.eat() and a.move() invoke the same methods for each different kind of animal - the Animal.eat and the Animal.move methods - but have different field values. Thus, the invocations of a.eat and a.move are not strictly polymorphism.