**Name:**

**Quiz#7 Generics**

class Animal {

public void draw(){}

}

class Dog extends Animal{}

class Test{

public void draw(List<Animal> list){

for(Animal a : list)

a.draw();

}

public static void main(String[] args){

Test t = new Test();

List<Dog> dogs = new ArrayList<Dog>();

t.draw(dogs);//ERROR WHY???

}

}

1. There is an error (as shown) in the code above. Why is there an error? What is the solution? <2 marks>

List<Dog> is NOT a subclass of List<Animal>. The danger would be if the method attempted to add an Animal object to the Dog list which may or may not be a Dog

Public void draw(List<? Extends Animal> list){

The above fixes the code and is allowed since we are not modifying the list passed in

class Groceries{}

class Fruit extends Groceries{}

class Apple extends Fruit{}

public void bagIt(List<Groceries> items, Groceries g){

items.add(g);

}

2. The function only works with actual Groceries and not it’s subclasses. The programmer would like it to work with Apples and Fruit as well as Groceries. Change the code so it will work and explain why it will work <2 marks>

Public void bagIt(List<? Super Apple> items, Groceries g){

This allows Apples and it’s super classes to be added to the list. It does allow Objects to be added as well

Class List<T> extends Collection<T>{}

Class ArrayList<T> extends List<T>{}

Class Payload<X,T> extends ArrayList<T>

3. Which of the following is NOT a subclass of Collection<String>? <1 mark>

Payload<String, String>

Payload<Integer,String>

**Payload<String,Integer>**

The generic T matches the 2nd param of Payload which is String from Collection thus it must be String in all subclasses

void swapFirst(List<? extends Number> l1, List<? extends Number> l2) {

Number temp = l1.get(0);

l1.set(0, l2.get(0));

l2.set(0, temp);}

4. The above code does not compile. Why is there a problem? Fix the code so it will work. <2 marks>

The issue is we cannot add to a wildcard list since l1 list could be Integer and l2 could be Double and swapping elements would result in an error in l1.

The solution is to return to standard generics as we would require both lists to be the same type (in the same fashion that the copy() method checks array types):

Void swapFirst(List<T> l1, List<T> l2)

Note that this also allows the function to be used for all types of lists, not just Number type lists