Quiz Section 9: Polymorphism – Solutions

Task 1 – We Game to Please [11 pts]

Suppose that we have the following classes:

```
class Game {..}
class CardGame extends Game {..}
class VideoGame extends Game {..}
class GoFish extends CardGame {..}
```

and the following variables:

```
Object obj;
Game game;                    List<Game> games;
CardGame cardGame;            List<? extends CardGame> cardGames;
VideoGame videoGame;          List<VideoGame> videoGames;
GoFish goFish;                List<? super GoFish> goFishGames;
```

For each line of code below, state whether the call is legal or illegal. If it is illegal, briefly explain why.

a) `cardGames.add(obj);`

This is illegal since `Object` is not a subclass of any subclass of `CardGame`. (`Object` is a superclass of `CardGame`)

b) `videoGames.add(obj);`

This is illegal since `Object` is not a subclass of `VideoGame`.

c) `goFishGames.add(obj);`

This is illegal since our list may be of type i.e., `List<Game>` and `Object` cannot be cast down into a game.

d) `goFishGames.add(goFish);`

This is legal since goFish is castable into any of its superclasses.

e) `cardGames.add(cardGame);`

This is illegal since `cardGames` could be, e.g., `List<GoFish>`, and `CardGame` is not a subclass of `GoFish`.

f) `goFishGames.add(null);`

This is legal.

g) `cardGame = cardGames.get(0);`

This is legal since anything extending `CardGame` is castable to it.

h) `videoGame = videoGames.get(0);`

This is legal.

i) `goFish = goFishGames.get(0);`

This is illegal since `goFishGames` could be, e.g., `List<Object>`.

j) `obj = goFishGames.get(0);`

This is legal since anything is castable to `Object`

k) `goFish = cardGames.get(0);`

This is illegal since `cardGames` could be, e.g., a `CardGame`.

Task 2 – A Compare to Remember [14 pts]

Answer each of the following questions about generic method declarations.

The first two parts use the type `Comparable<T>`, which is an interface containing only one method:

```
int compareTo(T other)
```

This function compares the object on which it is called, obj, and the passed-in object `other`. It returns -1 to indicate that obj comes before `other`, +1 to indicate that obj comes after `other`, and 0 if they can be placed in either order. (311 alert: the "before" relationship must be transitive, reflexive, and anti-symmetric.)

a) Consider the following alternative method declarations:

```
A: <E extends Comparable<E>> void randomize(Collection<E> vals)
B: <E extends Comparable<E>> void randomize(List<E> vals)
C: <E extends Comparable<? extends E>> void randomize(Collection<E> vals)
D: <E extends Comparable<? super E>> void randomize(List<E> vals)
```

Fill in the following table explaining the relationships between each pair of declarations. Write an "S" for if the declaration on left (the row) is stronger than the name on top (the column), a "W" if it is weaker, and a "—" if they are incomparable.

	A	**B**	**C**	**D**
A	X			
B		X		
C			X	
D				X

	A	**B**	**C**	**D**
A	X	S	W	—
B	W	X	W	W
C	S	S	X	—
D	—	S	—	X

b) The following type declaration is legal, but probably does not make sense. Why not?

```
Collection<? extends Comparable<?>>
```

This allows a collection of one type of object that are comparable to a different type of object. That means you can't actually compare elements of the collection to each other, e.g., for the purposes of sorting.

c) Consider the following alternative method declarations:

```
A: int find(Collection<?> sub, Collection<?> list)
B: int find(Collection<Object> sub, Collection<Object> list)
C: <T> int find(Collection<T> sub, Collection<T> list)
D: <T> int find(Collection<? extends T> sub, Collection<? super T> list)
```

Fill in the following table explaining the relationships between each pair of declarations. Write an "S" for if the declaration on left (the row) is stronger than the name on top (the column), a "W" if it is weaker, and a "—" if they are incomparable.

	A	**B**	**C**	**D**
A	X			
B		X		
C			X	
D				X

	A	**B**	**C**	**D**
A	X	S	S	S
B	W	X	W	W
C	W	S	X	W
D	W	S	S	X

Task 3 – Nothing Is Certain But Death and Maxes [9 pts]

Revisit your specification of `IntMaxStack` from Homework 5. Recall that this class implements the ordinary stack operations `push`, `pop`, and `size` but also adds the operation `max` that finds the largest element in constant time.

```
/**
 * Represents a **mutable** list of integers that can only be changed via
 * the stack operations, push and pop. It also provides a max operation that
 * returns the largest value in a non-empty stack.
 */
public interface IntMaxStack {
    /**
     * Returns the number of elements in the stack.
     * @return len(obj)
     */
    int size();

    /**
     * Returns the largest value in the stack, which must be non-empty.
     * @requires len(obj) != 0
     * @return max(obj), where
     *      max(x :: nil) := x
     *      max(x :: L) := x if x >= max(L)
     *      max(x :: L) := max(L) if x < max(L)
     */
    int max();

    /**
     * Adds the given value to the top (front) of the stack.
     * @param n the value that's to be added
     * @modifies obj
     * @effects obj = n :: obj_0
     */
    void push(int n);

    /**
    * Removes and returns the top element of the stack.
    * @requires len(obj) /= 0
    * @modifies obj
    * @effects obj_0 = n :: obj
    * @returns n
    */
    int pop()
}
```

In this problem, we will change it the interface to store data other than integers by introducing a type parameter `T` for the type of data stored in the stack.

a) Find all the places where "`int`" appears in your interface. Which of these should be replaced by `T`'s?

> The ones in `max`, `push`, and `pop` should be changed. The one in `size` should not be!

b) What bounds should we put on `T`?

> It should be `extends Comparable<T>` (or even more general `extends Comparable<? super T>`).

c) Next, we will make the change. Make `IntMaxStack` into a generic class `MaxStack` that can store other types of data.

```
public interface MaxStack<T extends Comparable<T>> {
    int size();

    T max();

    void push(T n);

    T pop();
}
```

d) Do the documentation/specifications still make sense? If not, what do we need to fix?

> The main interface description is specific to integers. It should be changed to say elements. The specification of `max` is also specific to integers. It simply needs a note saying that "$x < y$" means that `x.compareTo(y) < 0` and similarly for "$\geqslant$".