
1

Schreiben Sie ein generisches public-Interface X mit Typ-parameter T1

```
public interface X <                    > {

}
```

der auf A und die Subtypen von A beschränkt ist.

```
public interface X <T1 extends A> {

}
```

Das Interface X hat eine Objektmethode m1.
Der Rückgabetyp von m1 ist List (java.util.List) instanziiert mit C.

```
public interface X <T1 extends A> {
    List<C> m1 (                                    );
}
```

Methode m1 hat einen Parameter t vom formalen Typ T1 und einen Parameter lst von List.

```
public interface X <T1 extends A> {
    List<C> m1 ( T1 t, List<                > lst );
}
```

Methode m1 hat einen Parameter t vom formalen Typ T1 und einen Parameter lst von List.
Der generische Typparameter von lst ist auf B und alle Subtypen von B beschränkt.

```
public interface X <T1 extends A> {
    List<C> m1 ( T1 t, List<? extends B> lst );
}
```


2

Schreiben Sie eine generisches public-Klasse Z mit Typparameter T1, der auf A und die Subtypen von A beschränkt ist,

```
public class Z <T1 extends A>                    {

}
```

und X instanziiert mit T1 (aus dem 1. Teil) implementiert.

```
public class Z <T1 extends A> implements X<T1> {

}
```

Klasse Z soll ein protected-Attribut x vom Typ X mit Typparameter T1 haben.

```
protected X<T1> x;
```

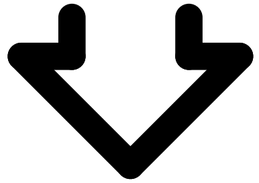
Der protected-Konstruktor hat einen Parameter x von Typ X mit Typparameter T1, mit dem das Objektattribut initialisiert wird.

```
protected X<T1> x;

protected Z (X<T1> x){
 this.x = x;
}
```

Der dynamische Typ der Rückgabe von Methode m1 soll List sein.

```
public interface X <T1 extends A> {
    List<C> m1 ( T1 t, List<? extends B> lst );
}
```



```
List<C> m1 ( T1 t, List<? extends B> lst ){

}
```

Falls lst mindestens drei Elemente hat, dann besteht die Rückgabe aus der Liste lst bei der die ersten beiden Elemente entfernt wurde. Andernfalls wird eine leere Liste zurückgegeben.

```
List<C> m1 ( T1 t, List<? extends B> lst ){
    List<C> result = new List<C>();
    if (lst.size() > 2)
        result = lst.subList(2, lst.size);
    return result;
}
```

Weiter soll Z eine public-Methode m3 definieren, aber nicht implementieren.
Diese hat einen Parameter a vom formalen Typ A und liefert nichts zurück.

```
public abstract void m3 (A a);
```

Weiter soll Z eine public-Methode m3 definieren, aber nicht implementieren.
Diese hat einen Parameter a vom formalen Typ A und liefert nichts zurück.

```
public abstract class Z <T1 extends A>
    implements X<T1> {

  public abstract void m3 (A a);

}
```

Komplettlösung

```
public interface X <T1 extends A> {
   List<C> m1 ( T1 t, List<? extends B> lst );
}

public abstract class Z<T1 extends A> implements X<T1>{

protected X<T1> x;

protected Z (X<T1> x){
 this.x = x;
}

public List<C> m1 ( T1 t, List<? extends B> lst ){
    List<C> result = new List<C>();
    if (lst.size() > 2)
        result = lst.subList(2, lst.size);
    return result;
}

public abstract void m3 (A a);
}
```