
Australian
National
University
Generics
J12

Our List Interface (Initial Attempt)

We will explore lists using a simple interface:

```
public interface ObjectList {
    void add(Object value);
    Object get(int index);
    int size();
    Object remove(int index);
    void reverse();
}
```

We can put anything in there, but we lose a lot of information!

Generics

Generics let us *parameterize* types and methods with *type variables*.

```
public interface ObjectList {
    void add(Object value);
    Object get(int index);
    int size();
    Object remove(int index);
    void reverse();
}
```

Generics

Generics let us *parameterize* types and methods with *type variables*.

```
public interface List<T> {
    void add(T value);
    T get(int index);
    int size();
    T remove(int index);
    void reverse();
}
```

Now we can use concrete versions of this interface:

```
List<Int>
List<String>
List<Person>
List<Animal>
```

…

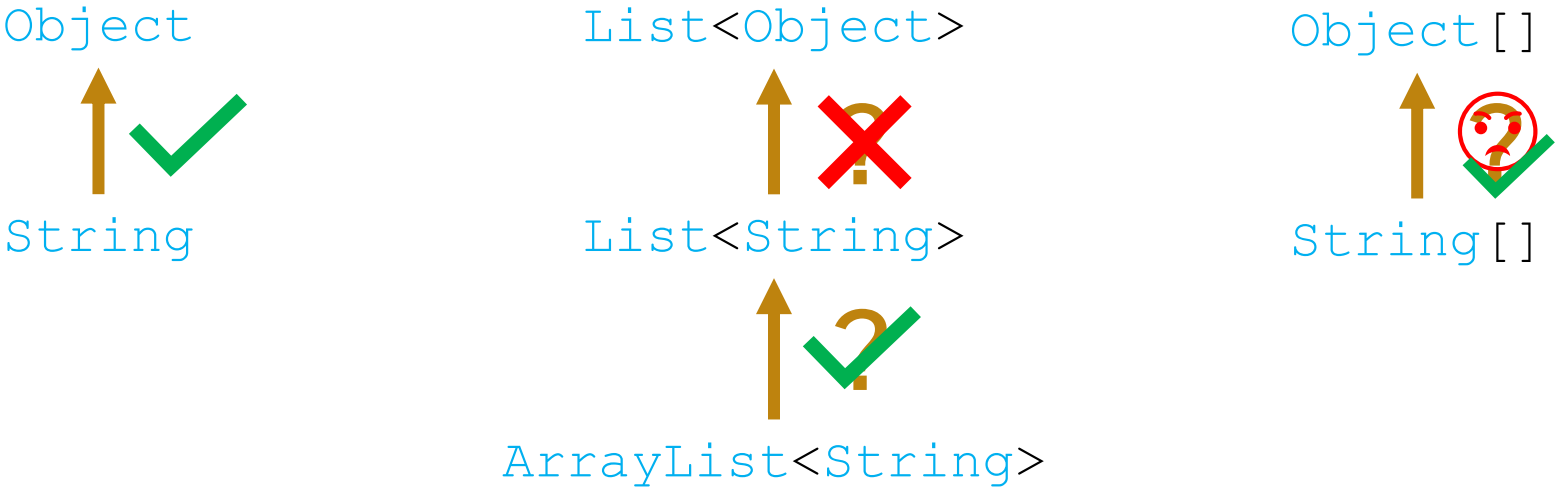
Same for generic classes.

Generic Methods

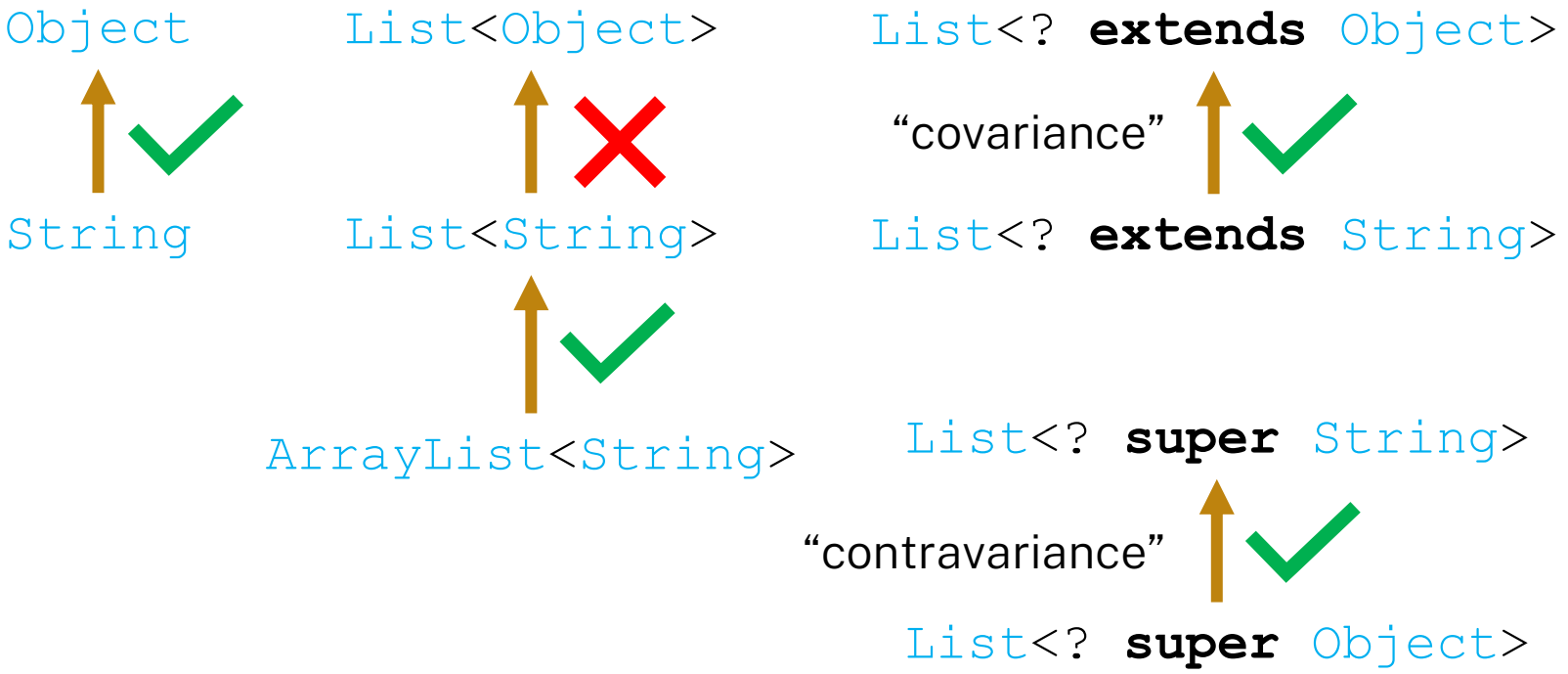
Sometimes we also want to be flexible on the level of methods.

```
public interface List<T> {
    …
    <R> List<R> map(Function<T,R> mapper);
}
```

Subtyping and Generics



Subtyping and Generics - Variance



Constraints

Lists normally only store and return their elements, so knowing nothing about `T` is fine. But sometimes you want to know more.

```
public interface SortedList<T extends Comparable<T>> {
    …
}
```

To be able to sort the list, we need to be able to compare its elements. `Comparable<T>` is a standard Java interface that prescribes a `compare` method and is implemented by `String`, `Int`, …

Java Generics - Restrictions

- No primitive types – use boxed versions
- No **`instanceof`** on type parameters
  - `x` **`instanceof`** `List<T>` becomes `x` **`instanceof`** `List`
- No access to type parameters `T` in static members