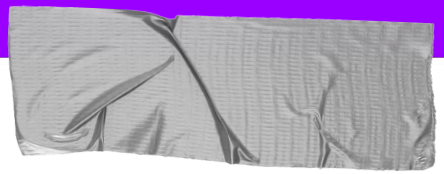
Java Certified #10

A question lead guide to prepare Java certification



Working with Arrays and Collections

Which methods compile?

- ➔ `public List<? super IOException> getListSuper() { return new ArrayList<Exception>(); }`
- ➔ `public List<? super IOException> getListSuper() { return new ArrayList<FileNotFoundException>();}`
- ➔ `public List<? extends IOException> getListExtends() { return new ArrayList<FileNotFoundException>();}`
- ➔ `public List<? extends IOException> getListExtends() { return new ArrayList<Exception>(); }`

public List<? super IOException> getListSuper() { return new ArrayList<Exception>(); }

public List<? extends IOException> getListExtends() { return new ArrayList<FileNotFoundException>();}

The methods that will compile are:

```
public List<? super IOException> getListSuper() {
        return new ArrayList<Exception>();
}
```

This method compiles because Exception is a super of IOException, and the list can contain IOException and its superclasses.

```
public List<? extends IOException> getListExtends() {
    return new ArrayList<FileNotFoundException>();
}
```

This method compiles because FileNotFoundException is a subclass of IOException, and the list can contain IOException and subclasses of IOException.

public List<? super IOException> getListSuper() { return new ArrayList<Exception>(); }

public List<? extends IOException> getListExtends() { return new ArrayList<FileNotFoundException>();}

The other methods will not compile because they attempt to return a list with a type incompatible with the bounded wildcard.

In

```
public List<? super IOException> getListSuper() {
    return new ArrayList<FileNotFoundException>();
}
```

FileNotFoundException is not a super class of IOException,

and in

```
public List<? extends IOException> getListExtends() {
    return new ArrayList<Exception>();
}
```

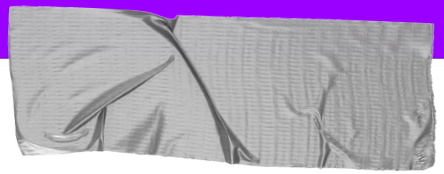
, Exception is not a subclass of IOException.

The wildcard bounds must be respected for the code to compile.

Bounded types in Java generics allow you to specify constraints on the types that can be used with a generic class or method.

They are denoted by `<? extends Type>` for upper bounds (*Type and subclasses of Type*) and `<? super Type>` for lower bounds (*Type and superclasses of Type*).

This ensures type safety and flexibility in your code.



Using Java I/O API

What is the output of the following snippet? (Assume the file exists)

```
Path path =  Paths.get("C:\\home\\joe\\foo");
System.out.println(path.getName( 0 ));
```

- **C**
- **C:**
- **home**
- **IllegalArgumentException**

home

getName(0) Returns the path element corresponding to the specified index.
The 0th element is the path element closest to the root.
https://docs.oracle.com/javase/tutorial/essential/io/pathOps.html
https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/nio/file/Path.html

JAVADOC
Path getName(int index)
Returns a name element of this path as a Path object.
The index parameter is the index of the name element to return. The element that is closest to the root in the directory hierarchy has index 0. The element that is farthest from the root has index count-1.
Parameters:
index - the index of the element
Returns:
the name element
Throws:
`IllegalArgumentException` - if index is negative, index is greater than or equal to the number of elements, or this path has zero name elements

Handling Date, Time, Text, Numeric and Boolean Values

Given:

```
public class FetchService {
    public static void main( String[] args ) throws Exception {
        FetchService service = new FetchService();
        String ack = service.fetch();
        LocalDate date = service.fetch();
        System.out.println( ack + " the " + date.toString() );    }

    public String fetch() {        return "ok";    }

    public LocalDate fetch() {        return LocalDate.now();    }
}
```

What will be the output?

- ok the 2024-07-10
- ok the 2024-07-10T07:17:45.523939600
- An exception is thrown
- Compilation fails

Compilation fails

Compilation fails because

`'fetch()'` clashes with `'fetch()';` both methods have same erasure

`'fetch()'` is already defined in `'com.vv.FetchService'`

The provided code will result in a compilation failure.

The reason is that the `FetchService` class has two methods named fetch with the same parameter list (i.e., no parameters).

In Java, method overloading requires the methods to have different parameter lists.

Since both fetch methods have identical signatures, this is not allowed.

https://bit.ly/javaOCP