

Java-Success.com

Industrial strength Java/JEE Career Companion for those who want to go places

[Home](#)
[Java FAQs](#)
[600+ Java Q&As](#)
[Career](#)
[Tutorials](#)
[Member](#)
[Why?](#)
[Can u Debug?](#)
[Java 8 ready?](#)
[Top X](#)
[Productivity Tools](#)
[Judging Experience?](#)

[Home](#) › [Interview](#) › [Core Java Interview Q&A](#) › [Generics](#) › 3 scenarios to get handle on Java generics

3 scenarios to get handle on Java generics

Posted on [November 22, 2014](#) by [Arulkumaran Kumaraswamipillai](#) — [No Comments](#) ↓

Q1. If `java.lang.Object` is the super type for `java.lang.Number` and, `Number` is the super type for `java.lang.Integer`, am I correct in saying that `List<Object>` is the super type for `List<number>` and, `List<Number>` is the super type for `List<Integer>`.

[9 tips to earn more](#) | [What can u do to go places?](#) | **945+** members. [LinkedIn Group](#). [Reviews](#)

600+ Full Stack Java/JEE Interview Q&As ♥Free ♦FAQs

[open all](#) | [close all](#)

[Ice Breaker Interview](#)

[Core Java Interview C](#)

[Java Overview \(4\)](#)

[Data types \(6\)](#)

[constructors-methc](#)

[Reserved Key Wor](#)

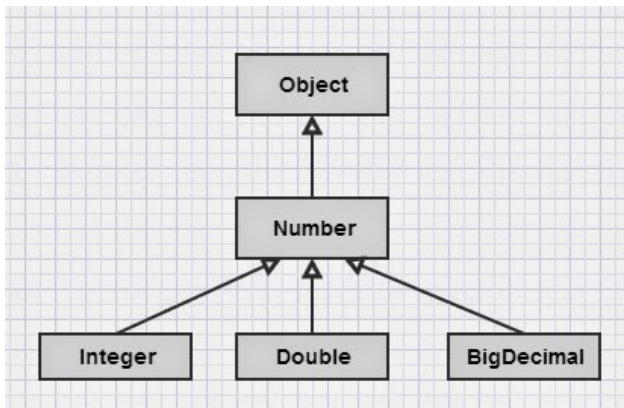
[Classes \(3\)](#)

[Objects \(8\)](#)

[OOP \(10\)](#)

[GC \(2\)](#)

[Generics \(5\)](#)



A1. No. `List<Object>` is not the the super type of `List<Number>`. If that were the case, then you could add objects of any type, and it defeats the purpose of Generics.

```

1  /* Compile Error: Type mismatch. Cannot convert
2  List<Number> numbers2 = new ArrayList<Integer>()
3
4
5  //Compiles
6  List<Integer> numbers3 = new ArrayList<Integer>()
7
8
9  /*Compile Error: Cannot instantiate the type Arr
10 List<? super Integer> numbers6 = new ArrayList<?
11
12
13 //Compiles
14 List<? super Integer> numbers7 = new ArrayList<I
15 numbers7.add(Integer.valueOf(5)); //ok
16
17
18 //Compiles
19 List<? extends Number> numbers5 = new ArrayList<
20 //Compile error: Read only.Can't add
21 numbers5.add(Integer.valueOf(5));
22
23
  
```

In Generics, wildcards (i.e. ?), makes it possible to work with **super classes** and **sub classes**.

Q2. How will you go about deciding which of the following to use?

- `<Number>`
- `<? extends Number>`
- `<? super Number>`

- ♥ Java Generics
- ♥ Overloaded m
- ♦ 12 Java Gener
- ♦ 7 rules to reme
- 3 scenarios to ge
- FP (8)
- IO (7)
- Multithreading (12)
- Algorithms (5)
- Annotations (2)
- Collection and Data
- Differences Between
- Event Driven Progr
- Exceptions (2)
- Java 7 (2)
- Java 8 (24)
- JVM (6)
- Reactive Programn
- Swing & AWT (2)
- JEE Interview Q&A (3)
- Pressed for time? Jav
- SQL, XML, UML, JSC
- Hadoop & BigData Int
- Java Architecture Inte
- Scala Interview Q&As
- Spring, Hibernate, & I
- Testing & Profiling/Sa
- Other Interview Q&A 1
- Free Java Interview

As a Java Architect

[Java architecture & design concepts](#)
[interview Q&As with diagrams](#) | [What should be a typical Java EE architecture?](#)

A2. Here is the guide:

1. Use the **? extends wildcard** if you need to retrieve object from a data structure. That is read only. You can't add elements to the collection.
2. Use the **? super wildcard** if you need to put objects in a data structure.
3. If you need to do both things (read and add elements), don't use any wildcard.

Let's take 3 scenarios to explain this.

Scenario 1: A custom generic class

GenericSingleTypeScenario class that handles only Integer types.

```

1 import java.util.List;
2
3 public class GenericSingleTypeScenario<Integer>
4
5     public void readOnly(List<? extends Number> num
6         for (Number number : numbers) {
7             System.out.println("readOnly: " + number);
8         }
9     }
10
11     public void writeOnly(List<? super Integer> num
12         numbers.add(aNumber);
13     }
14
15     public void writeAndRead(List<Integer> numbers,
16         numbers.add(aNumber);
17         for (Integer integer : numbers) {
18             System.out.println("readAndWrite: " + integer
19         }
20     }
21 }
22 }
23

```

Add a test class with a main method to test the above class

```

1 import java.util.ArrayList;
2 import java.util.List;
3
4 public class GenericSingleTypeScenarioTest {
5
6     public static void main(String[] args) {
7         GenericSingleTypeScenario<Integer> singleType
8         List<Integer> numbers = new ArrayList<Integer>
9         numbers.add(1); //autoboxes 1 to type Integer

```

Senior Java developers must have a good handle on

[open all](#) | [close all](#)

- [Best Practice \(6\)](#)
- [Coding \(26\)](#)
- [Concurrency \(6\)](#)
- [Design Concepts \(7\)](#)
- [Design Patterns \(11\)](#)
- [Exception Handling \(3\)](#)
- [Java Debugging \(21\)](#)
- [Judging Experience \(1\)](#)
- [Low Latency \(7\)](#)
- [Memory Management \(1\)](#)
- [Performance \(13\)](#)
- [QoS \(8\)](#)
- [Scalability \(4\)](#)
- [SDLC \(6\)](#)
- [Security \(13\)](#)
- [Transaction Management \(1\)](#)

80+ step by step Java Tutorials

[open all](#) | [close all](#)

- [Setting up Tutorial \(6\)](#)
- [Tutorial - Diagnosis \(2\)](#)
- [Akka Tutorial \(9\)](#)
- [Core Java Tutorials \(2\)](#)
- [Hadoop & Spark Tutorials \(1\)](#)
- [JEE Tutorials \(19\)](#)
- [Scala Tutorials \(1\)](#)
- [Spring & Hibernate Tutorials \(1\)](#)
- [Tools Tutorials \(19\)](#)
- [Other Tutorials \(45\)](#)

```

10
11 singleType.readOnly(numbers);
12 singleType.writeOnly(numbers, 6); //autoboxes
13 singleType.writeAndRead(numbers, Integer.valueOf(9));
14
15 }
16 }
17

```

Output:

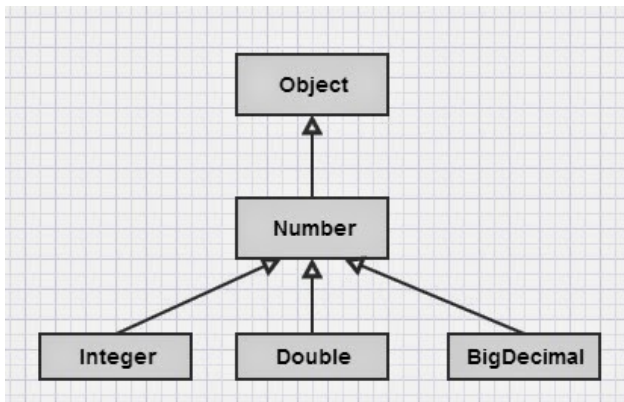
```

1 readOnly: 1
2 readAndWrite: 1
3 readAndWrite: 6
4 readAndWrite: 9
5

```

Scenario 2: A custom generic

class **GenericMultipleTypesScenario** class that handles *Integer* and *Double* types.



```

1 import java.util.List;
2
3 public class GenericMultipleTypesScenario<Number> {
4
5     public void readOnly(List<? extends Number> numbers) {
6         for (Number number : numbers) {
7             System.out.println("readOnly: " + number);
8         }
9     }
10
11     public void writeOnly(List<? super Number> numbers, Number aNumber) {
12         numbers.add(aNumber);
13     }
14
15     public void writeAndRead(List<Number> numbers, Number aNumber) {
16         numbers.add(aNumber);
17         for (Number number : numbers) {
18             System.out.println("readAndWrite: " + number);
19         }
20     }
21 }

```

Preparing for Java written & coding tests

[open all](#) | [close all](#)

- [◆ Complete the given](#)
- [Can you write code? \(1\)](#)
- [Converting from A to B](#)
- [Designing your classes](#)
- [Java Data Structures](#)
- [Passing the unit tests](#)
- [What is wrong with this](#)
- [Writing Code Home Assignment](#)
- [Written Test Core Java](#)
- [Written Test JEE \(1\)](#)

How good are your...to go places?

[open all](#) | [close all](#)

- [Career Making Knowledge](#)
- [Job Hunting & Resumes](#)

```
19    }
20
21    }
22 }
23
```

The test class

```
1 import java.util.ArrayList;
2 import java.util.List;
3
4 public class GenericMultipleTypesScenarioTest {
5
6     public static void main(String[] args) {
7         //Integer
8         GenericMultipleTypesScenario<Integer> integerT
9         List<Integer> numbers = new ArrayList<Integer>
10        numbers.add(1); //autoboxes 1 to type Integer
11
12        integerType.readOnly(numbers);
13        integerType.writeOnly(numbers, 6); //autoboxe
14        integerType.writeAndRead(numbers, Integer.valu
15
16        //Double
17        GenericMultipleTypesScenario<Double> doubleTyp
18        List<Double> numbersDouble = new ArrayList<Dou
19        numbersDouble.add(1.5); //autoboxes 1.5 to typ
20
21        doubleType.readOnly(numbersDouble);
22        doubleType.writeOnly(numbersDouble, 6.5); //a
23        doubleType.writeAndRead(numbersDouble, Double.
24
25    }
26 }
27
28
```

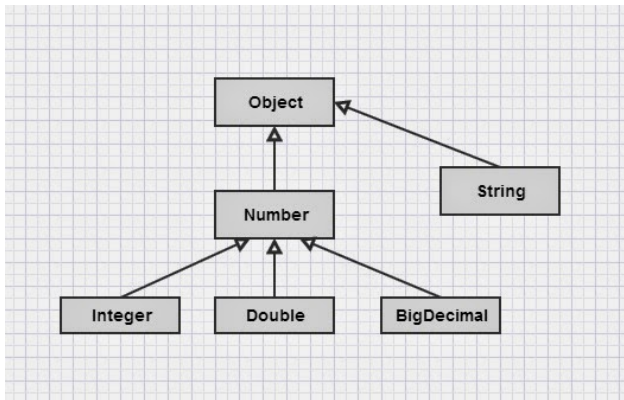
Output:

```
1 readOnly: 1
2 readAndWrite: 1
3 readAndWrite: 6
4 readAndWrite: 9
5 readOnly: 1.5
6 readAndWrite: 1.5
7 readAndWrite: 6.5
8 readAndWrite: 9.5
9
```

Scenario 3: A custom generic

class **GenericAnyTypesScenario** class that handles any

type like *Integer*, *Double*, *String*, etc. The any type is defined in this example as letter **T**. It can be any letter.



```

1  import java.util.List;
2
3  public class GenericAnyTypesScenario<T> {
4
5      public void readOnly(List<? extends T> values)
6          for (T value : values) {
7              System.out.println("readOnly: " + value);
8          }
9      }
10
11     public void writeOnly(List<? super T> values, T
12         values.add(aValue);
13     }
14
15     public void writeAndRead(List<T> values, T aValue) {
16         values.add(aValue);
17         for (T value : values) {
18             System.out.println("readAndWrite: " + value);
19         }
20     }
21 }
22 }
23
24

```

Test class

```

1  import java.util.ArrayList;
2  import java.util.List;
3
4  public class GenericAnyTypesScenarioTest {
5
6      public static void main(String[] args) {

```

```

7 //integer type
8 GenericAnyTypesScenario<Integer> integerType =
9 List<Integer> valuesIntegers = new ArrayList<I
10 valuesIntegers.add(1); //autoboxes 1 to type I
11
12 integerType.readOnly(valuesIntegers);
13 integerType.writeOnly(valuesIntegers, 6); //a
14 integerType.writeAndRead(valuesIntegers, Integ
15
16
17 //double type
18 GenericAnyTypesScenario<Double> doubleType = n
19 List<Double> valuesDouble = new ArrayList<Doub
20 valuesDouble.add(1.5); //autoboxes 1.5 to type
21
22 doubleType.readOnly(valuesDouble);
23 doubleType.writeOnly(valuesDouble, 6.5); //au
24 doubleType.writeAndRead(valuesDouble, Double.v
25
26
27 //string type
28 GenericAnyTypesScenario<String> stringType = n
29 List<String> valuesString = new ArrayList<Stri
30 valuesString.add("apple");
31
32 stringType.readOnly(valuesString);
33 stringType.writeOnly(valuesString, "orange");
34 stringType.writeAndRead(valuesString, "mango")
35
36 }
37 }
38
39

```

Output:

```

1  readOnly: 1
2  readAndWrite: 1
3  readAndWrite: 6
4  readAndWrite: 9
5  readOnly: 1.5
6  readAndWrite: 1.5
7  readAndWrite: 6.5
8  readAndWrite: 9.5
9  readOnly: apple
10 readAndWrite: apple
11 readAndWrite: orange
12 readAndWrite: mango
13

```

Popular Posts

♦ 11 Spring boot interview questions & answers

861 views

♦ Q11-Q23: Top 50+ Core on Java OOP Interview
Questions & Answers

829 views

18 Java scenarios based interview Questions and Answers

448 views

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

407 views

♦ 7 Java debugging interview questions & answers

311 views

♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

303 views

01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

294 views

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

288 views

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

263 views

8 Git Source control system interview questions & answers

215 views

Bio

Latest Posts



Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.



**About** [Arulkumaran Kumaraswamipillai](#)

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via [Amazon.com](#) in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.

◀ Java coding question on recursion and generics

Understanding Open/Closed Principle (OCP) from the SOLID OO principles with a Java example ▶

Posted in Generics, member-paid

Leave a Reply

Logged in as geethika. [Log out?](#)

Comment

Empowers you to open more doors, and fast-track

Technical Know Hows

☀ [Java generics in no time](#) ☀ [Top 6 tips to transforming your thinking from OOP to FP](#) ☀ [How does a HashMap internally work? What is a hashing function?](#)
☀ [10+ Java String class interview Q&As](#) ☀ [Java auto un/boxing benefits & caveats](#) ☀ [Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect](#)

Non-Technical Know Hows

☀ [6 Aspects that can motivate you to fast-track your career & go places](#) ☀ [Are you reinventing yourself as a Java developer?](#) ☀ [8 tips to safeguard your Java career against offshoring](#) ☀ [My top 5 career mistakes](#)

Prepare to succeed

☀ [Turn readers of your Java CV go from “Blah blah” to “Wow”?](#) ☀ [How to prepare for Java job interviews?](#) ☀ [16 Technical Key Areas](#) ☀ [How to choose from multiple Java job offers?](#)

Select Category ▼

© Disclaimer

The contents in this Java-Success are copy righted. The author has the right to correct or enhance the current content without any prior notice.

These are general advice only, and one needs to take his/her own circumstances into consideration. The author will not be held liable for any damages caused or alleged to be caused either directly or indirectly by these materials and resources. Any trademarked names or labels used in this blog remain the property of their respective trademark owners. No guarantees are made regarding the accuracy or usefulness of content, though I do make an effort to be accurate. Links to external sites do not imply endorsement of the linked-to sites.