Register | Login | Logout | Contact Us

Java-Success.com

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

search here … Go

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 › ♥ Java Generics in no time "? extends" & "? super" explained with a diagram

♥ Java Generics in no time "? extends" & "? super" explained with a diagram

Posted on June 25, 2015 by Arulkumaran Kumaraswamipillai

Generics in Java can be be a bit tricky to get your head around. Hope the explanation below enhances your understanding of generics. This complements 3 scenarios to get handle on Java generics.

Plain old List, List<Object>, and List<?>

The plain old List: is a heterogeneous mixture or a mixed bag that contains elements of all types, for example Integer, String, Pet, Dog, etc.

The List<Object>: is also a heterogeneous mixture like the plain old List, but not the same and can be more restrictive than a plain old List. It is incorrect to think of this as the super type for a collection of any object types.

The List<?>: is a homogenous collection that represents a family of generic instantiations of List like List<String>, List<Integer>, List<Pet>, List<Dog>, etc.

List<?> is the **super type** for all generic collection as Object[ ] is the super type for all arrays.

What can I assign to? What can I add to the Collection?

When working with Collection & Generics, you need to ask 4 important questions.

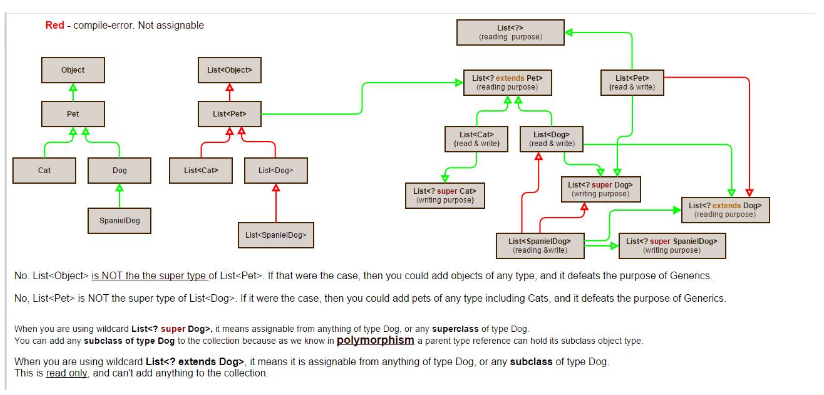
1) Can the RHS be assigned to the LHS?
2) What types of objects can I add to the collection?
3) Is it a read only or read & write collection?
4) When to use which wild card ("? extends", "? super") ?

If you do the wrong thing, you will get "compile-time" errors. Also, note that you can't use wildcards on the RHS when assigning and Java 8 supports empty "<>" on the RHS.

Understanding Generics and assign-abilities

As a Java Architect

Java architecture & design concepts interview Q&As with diagrams | What should be a typical Java EE architecture?



Red - compile-error. Not assignable
Object
Pet
Cat
Dog
SpanielDog
List<Object>
List<Pet>
List<Cat>
List<Dog>
List<SpanielDog>
List<?> (reading purpose)
List<? extends Pet> (reading purpose)
List<Pet> (read & write)
List<Cat> (read & write)
List<Dog> (read & write)
List<? super Cat> (writing purpose)
List<? super Dog> (writing purpose)
List<? extends Dog> (reading purpose)
List<SpanielDog> (reading &write)
List<? super SpanielDog> (writing purpose)

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

No, List<Pet> is NOT the super type of List<Dog>. If it were the case, then you could add pets of any type including Cats, and it defeats the purpose of Generics.

When you are using wildcard **List<? super Dog>**, it means assignable from anything of type Dog, or any **superclass** of type Dog.
You can add any **subclass of type Dog** to the collection because as we know in **polymorphism** a parent type reference can hold its subclass object type.

When you are using wildcard **List<? extends Dog>**, it means it is assignable from anything of type Dog, or any **subclass** of type Dog.
This is read only, and can't add anything to the collection.

Java Generics Overview for assignability

Note: "**?**" is a wild card meaning anything. "? extends Pet" means "**anything** that extends a Pet".

Now let's see a code example based on the above diagram. Click on the diagram to expand.

```
package com.generics;

import java.util.ArrayList;
import java.util.List;

public class GenericsAssignable {

    public static void main(String[] args) {
        new GenericsAssignable().create();
    }

    public void create() {

        List<Object> objectsBad = new ArrayList<
        List<Pet> petsBad = new ArrayList<Dog>()


        //===== "?" and "? extends" - read only
        List<?> petsOk = new ArrayList<Pet>(); /
        List<? extends Pet> petsOk2 = new ArrayL
        List<? extends Pet> petsOk3 = new ArrayL
        List<? extends Dog> petsOk4 =  new Array
        List<? extends Dog> petsOk5 =  new Array

        List<? extends Dog> petsBad2 =  new Arra

        //====  "? super" - can add objects to c
        List<? super Dog> petsOk6 = new ArrayLis
        List<? super Dog> petsOk7 = new ArrayLis
        List<? super Dog> petsOk8 = new ArrayLis

        //can add Dog or any subclass of Dog
        petsOk6.add(new Dog());
        petsOk6.add(new SpanielDog());//polymorp

        petsOk6.add(new Pet()); //4. COMPILE ERR
```

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 I
⊞ Low Latency (7)
⊞ Memory Managemen
⊞ Performance (13)
⊞ QoS (8)
⊞ Scalability (4)
⊞ SDLC (6)
⊞ Security (13)
⊞ Transaction Managen

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 Tuto
⊞ JEE Tutorials (19)
⊞ Scala Tutorials (1)
⊞ Spring & HIbernate Tu
⊞ Tools Tutorials (19)
⊞ Other Tutorials (45)

```

        List<? super Dog> petsBad3 = new ArrayLi

    }
}
```

5 Compile Errors marked in the above code reasoning

#1. List<Pet> is NOT the super type of List<Dog>. If it were the case, then you could add pets of any type including Cats, and it defeats the purpose of Generics. **List<?>** is the super type of **List<Pet>**. But read only. Can't add any objects.

#2. Same as **#1**. If were not illegal, you could add a Cat to a Dog collection. Defeating the purpose of Generics.

#3. List<? extends Dog> means assignable from any objects that are of type **Dog** or **subclasses** of Dog. Pet is a **superclass** of Dog.

#4. You can only add objects of type **Dog** or **subclasses** of Dog. Subclasses are possible because of **polymorphism** , where a parent type reference can hold its subclass object type. So, you can add objects of type Dog or its subclasses like SpanielDog, but NOT its super classes.

#5. List<? super Dog> means assignable from any objects that are of type **Dog** or **superclasses** of type Dog. Pet, Object, etc are valid superclasses. But SpanielDog is a subclass.

So, how to decide when to use a wild card, and when not to?

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 add objects to a data structure.

3. If you need to do both things (i.e. read and add objects), don’t use any wildcard.

More code examples to validate your understanding

```java
package com.generics;

import java.util.ArrayList;
import java.util.List;

public class GenericsByExample {

    public static void main(String[] args) {
        new GenericsByExample().create();
    }

    public void create() {

        // dogs only
        List<Dog> dogs = new ArrayList<Dog>();
        dogs.add(new Dog());
        dogs.add(new SpanielDog()); //polymorphi
        dogs.add(new Pet()); // compile error
        dogs.add(new Cat()); // compile error

        // cats only
        List<Cat> cats = new ArrayList<Cat>();
        cats.add(new Cat());
        // cats.add(new Dog()); //No can't add d

        // any pets can be added
        List<Pet> pets = new ArrayList<Pet>();
        pets.add(new Dog()); //polymorphism
        pets.add(new Cat()); //polymorphism
        pets.add(new Pet());

        // so, wrong to say List<Pet> is a super
        // defeats the purpose of having Generic

        // RHS allows Pet, Dog, Cat, SpanielDog,
        List<? extends Pet> petsOnlyForReading =

        // RHS allows Dog and SpanielDog, but r
        List<? extends Dog> dogsOnlyForReadingy

        petsOnlyForReading.add(new Dog()); // co
        dogsOnlyForReadingy.add(new Cat()); // c

        List<Dog> dogsOnly = new ArrayList<Pet>(

        List<? super Dog> dogsOnly1 = new ArrayL
        dogsOnly1.add(new Dog());
        dogsOnly1.add(new SpanielDog()); // poly
        dogsOnly1.add(new Pet()); // compile err

        List<? super Dog> dogsOnly2 = new ArrayL
        dogsOnly2.add(new Dog());
        dogsOnly2.add(new SpanielDog()); // poly
        dogsOnly2.add(new Pet()); // compile err

        List<? super Dog> dogsOnly3 = new ArrayL
        dogsOnly3.add(new Dog());
        dogsOnly3.add(new SpanielDog()); // poly
        dogsOnly3.add(new Pet()); // compile err
        dogsOnly3.add(new Object()); // compile

        List<? super SpanielDog> spanielsOrSubcl
        spanielsOrSubclassesOnly.add(new Dog());
        spanielsOrSubclassesOnly.add(new Spaniel
        spanielsOrSubclassesOnly.add(new Pet());
    }
}
```

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.

‹ Remote debugging in Java with Java Debug Wire Protocol (JDWP)

03: ♦ ♥ Java autoboxing & unboxing – benefits & caveats interview Q&A ›

Posted in Generics

Tags: Free Content, Free FAQs

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