Difference between List, List<?>, and List<Object> in Java

A popular Java interview question based on Generics and Collection

JAVINPAUL

MAR 15, 2024 • PAID

Hello guys, since you guys like **How ConcurrentHashMap works in Java** and **Why String is Immutable in Java**, I am back with another core Java interview question for this week.

What is the difference between List, List<?>, and List<Object> in Java?

This is another tricky question that is commonly asked to Java programmers with few years of experience to test their knowledge about Generics and Collections. While they look really similar, there are subtle differences between them and it requires good knowledge of Java to answer this question.

By the way, if you like to receive Java interview questions twice a week in your email, please consider getting a paid subscription, it's just $5 per month, thank you for your support.

Coming back to topic, In Java, `List`, `List<?>`, and `List<Object>` represent different concepts and have different meanings:

1. `List`
   This refers to a **raw type of a list**. It's a generic interface that represents an ordered collection of elements. However, when you use `List` without specifying a type parameter (e.g., `List<Integer>`), you lose type safety.

   This means you can add any type of object to the list without the compiler detecting potential type errors. Raw types are discouraged in modern Java programming because they can lead to bugs that are harder to detect at compile time.

2. `List<?>`
   This is a wildcard type. It represents a list of some unknown type. It can be thought of as a list of elements of any type. It's similar to saying "**I don't care what the actual type of the elements is, as long as they're all the same type.**"

   This is useful when you want to work with a list in a generic way without caring about the

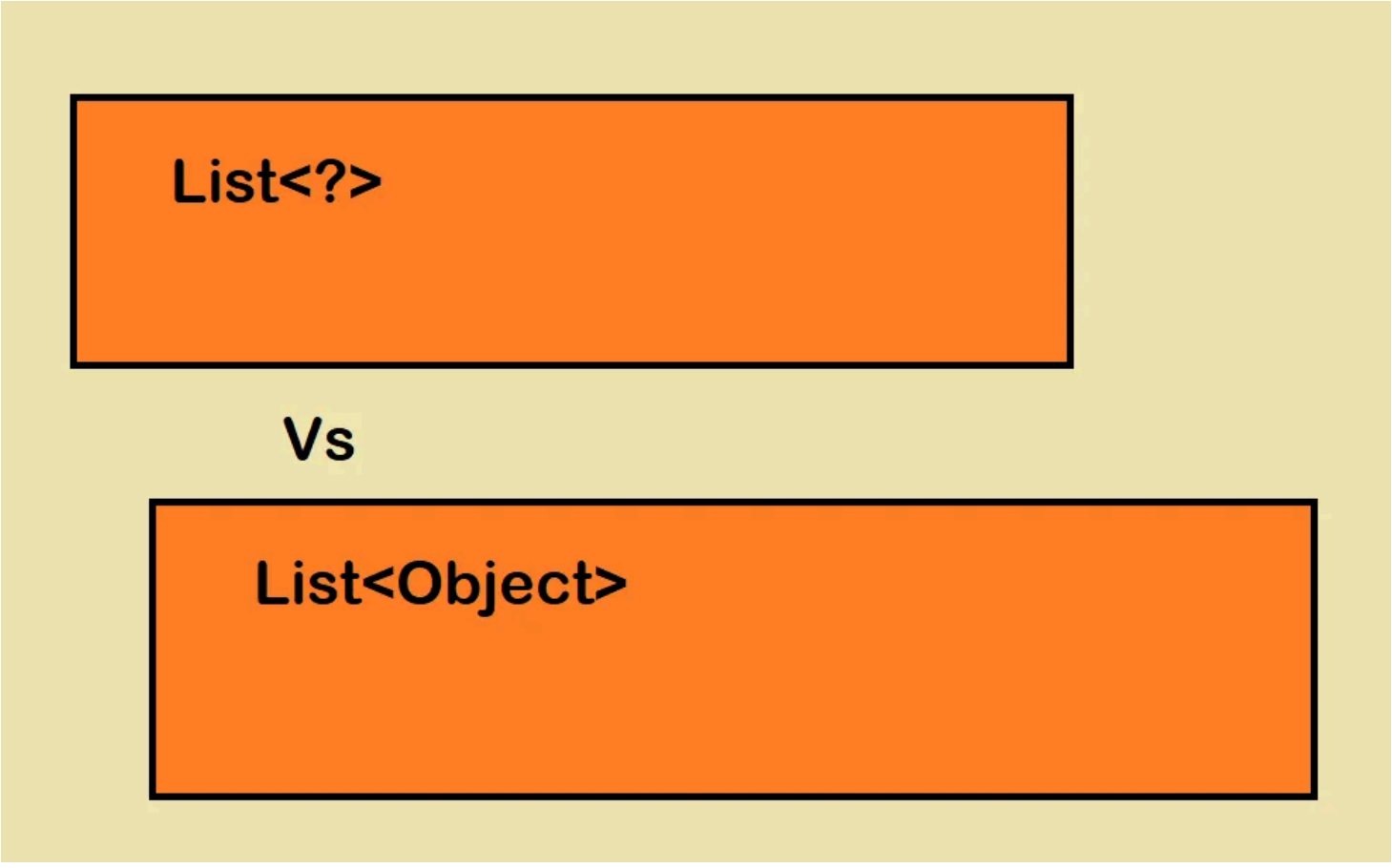
specific type of its elements. For example, if you have a method that only needs to iterate over the elements of a list without knowing their type, you can use `List<?>`.

3. `List<Object>`: This represents a list of objects where `Object` is the most generic type in Java. It means that the list can contain objects of any type because every class in Java is a subclass of `Object`. However, when you use `List<Object>`, you lose type specificity.

   For example, if you add a `String` to a `List<Object>`, you can retrieve it from the list as an `Object`, but you'd need to explicitly cast it back to a `String` if you want to use `String`-specific methods. This can lead to runtime errors if you make incorrect assumptions about the types stored in the list.

In summary:

- Use `List` with caution, as it represents a raw type lacking type safety.
- Use `List<?>` when you want to work with a list in a generic way without caring about the specific type of its elements.
- Use `List<Object>` when you specifically want a list that can contain objects of any type, but be cautious about the loss of type specificity and potential runtime errors.



Let's see examples for each type of list:

`List` (Raw Type)

Here is how we use a raw List, this is long back, prior to Java 5, in the early 2000s

```
List myList = new ArrayList(); // Using raw type
myList.add("Hello"); // Adding a String
myList.add(123); // Adding an Integer (allowed due to raw type)
String firstElement = (String) myList.get(0); // Need to cast back to
String
System.out.println(firstElement.toUpperCase()); // No compile-time
error, but may cause ClassCastException at runtime
```

In this example, we're using the raw type `List`. We can add elements of any type to the list, but we lose type safety and need to perform explicit casts when retrieving elements.

`List<?>` (Wildcard Type)

This is how we use List of unknown type in Java:

```
List<?> myList = new ArrayList<String>(); // Using wildcard type
myList.add("Hello"); // Compile-time error: Cannot add anything to a
wildcard list
Object firstElement = myList.get(0); // Allowed, but the type is Object
System.out.println(firstElement); // No need for casting, but can only
use Object methods
```

In this example, we're using a wildcard type `List<?>`. We can't add anything to this list (except `null`), but we can retrieve elements without needing to cast them. However, since the type is unknown, we can only treat retrieved elements as `Object`.

`List<Object>`

Here is an example of List<Object> in Java:

```
List<Object> myList = new ArrayList<>(); // Using List<Object>
myList.add("Hello"); // Adding a String
myList.add(123); // Adding an Integer
String firstElement = (String) myList.get(0); // Need to cast back to
String
System.out.println(firstElement.toUpperCase()); // No compile-time
error, but may cause ClassCastException at runtime
```

In this example, we're using `List<Object>`, which allows us to add objects of any type. We can retrieve elements without casting, but we might need to cast them if we want to use methods specific to their actual types.

These examples demonstrate the differences between `List`, `List<?>`, and `List<Object>` in Java.

Which one are you supposed to use?

The choice of which type of list to use depends on the specific requirements and design of your program:

1. **Use `List<E>` where possible**: It's generally recommended to use the generic type `List<E>` (where `E` is a type parameter representing the element type) whenever you have specific knowledge about the type of elements you'll be working with. This provides type safety and allows the compiler to catch potential errors at compile time.
2. **Use `List<?>` when you need to work with unknown types**: If you're writing code that needs to handle lists of unknown types in a generic way, you can use `List<?>`. This is particularly useful when you're designing methods or classes that operate on lists without needing to know the specific type of elements.
3. **Use `List<Object>` sparingly**: Using `List<Object>` can be appropriate when you need to deal with heterogeneous collections where you truly need to allow any type of object to be stored in the list. However, you should be cautious because it can lead to loss of type safety and potential runtime errors due to the need for explicit casting.

In summary, prefer to use generic types (`List<E>`) when you have specific knowledge about the types of elements you're working with. Use wildcard types (`List<?>`) when you need to work with lists of unknown types in a generic way. Reserve the use of `List<Object>` for scenarios where you truly need to handle heterogeneous collections, but be mindful of its limitations and potential for runtime errors.

That’s all guys, all the best for your interviews !!

And, if need help, you can also check my **Java + Spring Interview + SQL Bundle on Gumroad**, use code **friends20** to get a 20% discount also

← Previous

Next →

Comments

Write a comment...