Wildcard Parameter Types (bounded & unbounded)

Wildcards in generics are used to represent unknown types, allowing for more flexible and reusable code. They come in two main varieties: bounded and unbounded wildcards. Let's break down both concepts with simple explanations and examples.

1. Unbounded Wildcards (<?>)

An unbounded wildcard represents an unknown type. It allows a method or class to work with different types of objects without knowing the specific type in advance.

Example:

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

public class UnboundedWildcardExample {
    public static void printList(List<?> list) {
        for (Object item : list) {
            System.out.println(item);
        }
    }

    public static void main(String[] args) {
        List<Integer> intList = new ArrayList<>();
        intList.add(1);
        intList.add(2);

        List<String> strList = new ArrayList<>();
        strList.add("Hello");
        strList.add("World");

        printList(intList);  // Works with List<Integer>
        printList(strList);  // Works with List<String>
    }
}
```

Explanation:

- **Unbounded Wildcard (<?>)**: List<?> means a list of some unknown type. The printList method can accept a list of any type, but it can only work with the list elements as Object, since the specific type is not known.
- **Flexibility**: This is useful when you want to write code that can handle various types in a generic way.

Benefits of Using Generics

1. **Type Safety:** The compiler enforces type constraints, reducing the risk of ClassCastException at runtime.
2. **Code Clarity:** Generics make your code clearer by explicitly specifying the types being used.
3. **Avoids Unnecessary Casting:** With generics, you don't need to cast elements when retrieving them from a collection.

In summary, while the original code with raw types will work, its better practice to use generics (e.g., List<?>) to take advantage of compile-time type checking and ensure type safety.

2. Bounded Wildcards

Bounded wildcards allow you to specify an upper or lower limit on the type that the wildcard can represent. This ensures type safety and provides more control over the types that can be used.

Upper Bounded Wildcards (<? extends T>)

An upper bounded wildcard represents a type that is a subtype of a specified type T. It is useful when you want to work with a group of related types that extend a certain base type.

Example:

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

public class UpperBoundedWildcardExample {
    public static void printNumbers(List<? extends Number> list) {
        for (Number number : list) {
```

```
            System.out.println(number);
        }
    }

    public static void main(String[] args) {
        List<Integer> intList = new ArrayList<>();
        intList.add(1);
        intList.add(2);

        List<Double> doubleList = new ArrayList<>();
        doubleList.add(1.1);
        doubleList.add(2.2);

        printNumbers(intList);   // Works with List<Integer>
        printNumbers(doubleList); // Works with List<Double>
    }
}
```

Explanation:

- **Upper Bounded Wildcard (<? extends Number>)**: List<? extends Number> means a list of some type that extends Number. You can pass lists of Integer, Double, or any other type that extends Number to the printNumbers method.
- **Use Case**: It's used when you want to read items from a collection, and you want to ensure that the collection contains elements that are of type Number or its subtypes.

b. Lower Bounded Wildcards (<? super T>)

A lower bounded wildcard represents a type that is a supertype of a specified type T. It is useful when you want to work with a group of related types that are superclasses of a certain type.

Example:

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

public class LowerBoundedWildcardExample {
    public static void addNumbers(List<? super Integer> list) {
```

```
        list.add(1);
        list.add(2);
    }

    public static void main(String[] args) {
        List<Number> numberList = new ArrayList<>();
        addNumbers(numberList); // Works with List<Number>

        List<Object> objectList = new ArrayList<>();
        addNumbers(objectList); // Works with List<Object>

        System.out.println(numberList);
        System.out.println(objectList);
    }
}
```

Explanation:

- **Lower Bounded Wildcard (<? super Integer>)**: List<? super Integer> means a list of some type that is a supertype of Integer. This could be a List<Number> or List<Object>.
- **Use Case**: It's used when you want to add items to a collection, and you want to ensure that the collection can accept items of type Integer or its subtypes.

Summary

- **Unbounded Wildcard (<?>)**: Represents an unknown type. Used when you don't need to know the exact type.
- **Upper Bounded Wildcard (<? extends T>)**: Represents a type that is a subtype of T. Used for reading from a collection.
- **Lower Bounded Wildcard (<? super T>)**: Represents a type that is a supertype of T. Used for writing to a collection.

These wildcards help in writing flexible and reusable code by allowing you to specify and control the types of objects that can be used in generic classes and methods.