Unbounded Wildcards

03/07/2024

* An unbounded wildcard is represented by <?>. it means the data type is unknown.
* For instance, consider a method that prints the contents of a list:

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

Bounded Wild cards:

* There are instances where you might want to restrict the types that a method accepts. This is where bounded wildcards come into play.
* Upper Bounded Wildcards use the extends keyword.
* For example, a method processing numbers might only want to accept lists of number or its subclasses.

```
public void processNumbers(List<? extends Number> numbers
{
// process the list here
}
```

Lower Bound

* Lower Bounded wildcards use the super keyword.
* They restrict the unknown type to be a particular type or a superclass of that type.

```
public void addIntegers(List<? super Integer> list)
{ list.add(new Integer(50)); // adds the integer to the list
}
```

Functional Interface

* An interface is declared with only one abstract method, then it is referred as Functional Interface.
* The method in functional interface is called as functional method.

Along with the functional method, you can also add default and static method to functional interface.

ex of Function interfaces are:
* Runnable
* ActionListener
* ItemListener

Lambda Expression

* A lambda expression is an anonymous block of code that encapsulates an expression or a series of statements and returns a result.
* Syntax: (parameters) → { statements; }
* Lambda expression allows for creation and use of single method anonymous classes instead of creating separate concrete class for functional interface.
* They can accept zero or more parameters; any of which can be passed with or without type specification.
* Then signature of lambda expression implementing Runnable interface will be
() → void.

uses of Lambda Expressions.

* Lambda expressions are an important addition to Java that greatly improves overall maintainability, readability and developer productivity.
* They can be applied in many different contexts, ranging from simple anonymous functions to sorting and filtering collections.

* Lambda expressions can be assigned to variables and then passed into other objects
* It allows developers to write cleaner, shorter more readable and reusable code.

Functional Interface: Implementation:

Lambda Expression:

```
public class MaxFinderImpl Implements MaxFinder{
    @ override
    public int maximum(int num1, int num2) {
        return num1>num2?num1:num2;
    }
}
```

```
MaxFinder finder =(num1,num2) ->
    num1>num2?num1:num2;
int result = finder.maximum(10,20);
```

Return type of "λE" is functional Interface.