

CSE12 - Lecture 6 - B00

Wednesday, October 5, 2022 9:00 AM

PA1 hard deadline tonight
PA2 released today → due Tuesday
Discussion 4, 7, 8

Java Generics

`public interface Collection<E> extends Iterable<E>`

What does the <E> mean in the above code?

- A. That this collection can only be used with objects of a built-in Java type called E
- ☒ B. That an object reference that implements Collection can be instantiated to work with (almost) any object type
- C. That a single collection can hold objects of different types

parameterized type

object/reference types
does not work with primitives

no int
double

use Integer
Double

Java Generics use parameterized types in class definitions

```
public class RecentRememberer<T> {  
    private ArrayList<T> elements;
```

```
    public RecentRememberer() {
```

`elements = new ArrayList<T>();`

```
    }  
    public T add(T element) {
```

`elements.add(element);`
`return element;`

```
    }  
    public int getNumElements() {
```

`return elements.size();`

```
    }  
    public T getLastElement() {
```

`return elements.get(elements.size() - 1);`

```
    }  
}
```

What is the type parameter for the RecentRememberer class?

T

Complete the implementation of the RecentRememberer class.

Complete the following main method to create an instance of rr for integers and rr2 for strings.

```
public static void main(String[] args) {
```

`RecentRememberer<Integer> rr = new RecentRememberer<Integer>();`

`RecentRememberer<String> rr2 = new RecentRememberer<>();`

```
    rr.add(1);  
    rr.add(2);  
    rr2.add("three");  
    System.out.println(rr.getNumElements() + "elems added");  
    System.out.println("Last elem was " + rr.getLastElement());  
}
```

What gets printed?

2 elems added
Last element was 2

The type parameter can be used to stand for a type (to be specified later anywhere in this class (and its inner classes!))

You are not allowed to use Generics as follows:

In creating an object of that type:

```
new T() // error
```

In creating an array with elements of that type:

```
new T[100] // error
```

As an argument to instanceof:

```
someref instanceof T // error
```

Note: To ensure that certain methods can be called, we can constrain the generic type to be subclass of an interface or class

```
public class MyGenerics <E extends Comparable>{ .....}
```

↳ compare To()

Generics - <https://docs.oracle.com/javase/tutorial/java/generics/erasure.html>

Important for data structures in general

```
public class MyList<E>{  
    //codes that use E  
}
```

Pros of using generics

- Avoid type casting (i.e. limit runtime errors)

Before Java 5

```
ArrayList list = new ArrayList();// a list of objects  
list.add("greg")  
list.add(new Integer(12));  
  
Integer data = list.get(1);
```

Cons of using generics

- Type erasure

Type erasure during compile time

- Compiler checks if generic type is used properly. Then replace them with Object
- Runtime doesn't have different generic types

```
MyList<String> ref1 = new MyList<String>();  
MyList<Integer> ref2 = new MyList<Integer>();
```

Compile time:

```
MyList<String> ref1 = new MyList<String>();
```

Runtime

```
MyList<Object> ref1 = new MyList<Object>();
```

$E[] \text{ arr} = (E[] \text{ new Object}[])$

Convert Node and LinkedList to be a generic using List interface

```
public interface List<Element> {  
    /* Add an element at the end of the list */  
    void add(Element s);  
    /* Get the element at the given index */  
    Element get(int index);  
    /* Get the number of elements in the list */  
    int size();  
}  
  
class Node {  
    String value;  
    Node next;  
    public Node(String value, Node next) {  
        this.value = value;  
        this.next = next;  
    }  
}  
  
public class LinkedList implements StringList {  
    Node front;  
    int size;  
  
    public LinkedList() {  
        this.front = new Node(null, null);  
        this.size = 0;  
    }  
    public String get(int index) {  
        Node temp = this.front.next;  
        for (int i = 0; i < index; i += 1) {  
            temp = temp.next;  
        }  
        return temp.value;  
    }  
    public int size() {  
        return this.size;  
    }  
    public void add(String s) {  
        Node temp = this.front;  
        while (temp.next != null) {  
            temp = temp.next;  
        }  
        temp.next = new Node(s, null);  
        this.size += 1;  
    }  
}
```

Exceptions

What happens if an invalid index is passed to get()?

Null exception

Modify get() to throw an exception if the index is invalid

```
public String get(int index) {  
    Node temp = this.front.next;  
    for (int i = 0; i < index; i += 1) {  
        temp = temp.next;  
    }  
    return temp.value;  
}
```

*if (index < 0 ||
index >= size) ?*

*throw new IndexOutOfBoundsException();
new IllegalArgumentException();*

3

jUnit - test that an exception is thrown

[@Test(expected = IndexOutOfBoundsException.class)

Test fails if no IOBE exception is thrown

Write a test to verify get() throws an exception with an invalid index