

CSE 12 – Basic Data Structures and Object-Oriented Design

Lecture 6

Greg Miranda & Paul Cao, Winter 2021

Announcements

- Quiz 6 due Wednesday @ 8am
- Survey 2 due tonight @ 11:59pm
- PA2 released yesterday – closed PA
- Monday – Holiday
 - No class
 - No quiz due

Topics

- Generics & Exception Exercises

Generics and Exceptions

Java Generics



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

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

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

Java Generics

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 )  
    {  
        ...  
    }  
    ...
```

Type parameter

Element

E

T

K

V

Java Generics

Java Generics use parameterized types in class definitions

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

```
    public RecentRememberer() {...}  
    public T add(T element) {...}  
    public int getNumElements() {...}  
    public T getLastElement() {...}
```

```
}
```



Type parameter

Java Generics

Integer
String

```
public class RecentRememberer<T> {
```

```
private ArrayList<T> elements;
```

Is this line legal Java code?

61 ☒ A. Yes
15 B. No

```
public RecentRememberer()
```

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

String

```
public T add( T element )
```

```
{ ... String
```

```
}
```

```
...
```

String

ArrayList<String>

Java Generics

```
public class RecentRememberer<T> {  
  
    private ArrayList<T> elements;  
  
    public RecentRememberer()  
    {  
        elements = new ArrayList<T>();  
    }  
  
    public T add( T element )  
    { ...  
    }  
    ...  
}
```

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

Java Generics



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

```
    public RecentRememberer() {...}  
    public T add(T element) {...}  
    public int getNumElements() {...}  
    public T getLastElement() {...}
```

Will the main method compile?

21 A. Yes

55 **B. No**

```
public static void main(String[] args) {  
    RecentRememberer<T> rr = new RecentRememberer<T>();  
    RecentRememberer<T> rr2 = new RecentRememberer<T>();  
    rr.add(1);  
    rr.add(2);  
    rr2.add("three");  
    System.out.println(rr.getNumElements() + "elems added");  
    System.out.println("Last elem was " + rr.getLastElement());  
}
```

Java Generics

String
Integer

```
public class RecentRememberer<T> {  
    private ArrayList<T> elements = new ArrayList<T>();  
  
    public RecentRememberer() {...}  
    public T add(T element) {...}  
    public int getNumElements() {...}  
    public T getLastElement() {...}
```

Will the main method compile?

71 A. Yes
8 B. No

```
    public static void main(String[] args) {  
        RecentRememberer<Integer> rr = new RecentRememberer<Integer>();  
        RecentRememberer<String> rr2 = new RecentRememberer<String>();  
        rr.add(1);  
        rr.add(2);  
        rr2.add("three");  
        System.out.println(rr.getNumElements() + "elems added");  
        System.out.println("Last elem was " + rr.getLastElement());  
    }  
}
```

A few Notes

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>{}

MyGenerics <String>
MyGenerics <Student> → implement Comparable

Some quick words on Generics

- Important for data structures in general

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

- 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> . . .

- Runtime

MyList<Object>

More words on generics

- Pro


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

Before Java 5

```
ArrayList list = new ArrayList(); // a list of objects  
list.add("paul")  
list.add(new Integer(12));
```

```
Integer data = list.get(1);
```

- Con

- Type erasure 

Generics

- Convert LinkedList to be a generic

```

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 LinkedListStringList implements StringList {
    Node front;
    int size;

    public LinkedListStringList() {
        this.front = new Node(null, null);
        this.size = 0;
    }

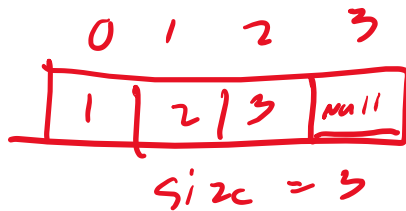
    public String get(int index) {
        Node temp = this.front.next;
        for (int i = 0; i < index; i += 1) {
            temp = temp.next;
        }
        return temp.value;
    }

    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;
    }
}

```

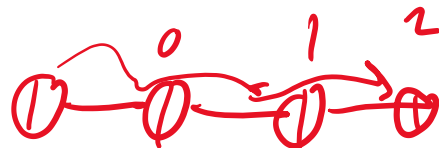

Exceptions



get(3)

- What happens if an invalid index is passed to get()?
- 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;  
}
```



size = 3

get(3)

↳ Null Pointer Exception

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

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class TestList {

    @Test(expected = IndexOutOfBoundsException.class)
    public void testNegativeIndex() {
        List<String> slist = new ArrayList<String>();
        slist.add("banana");
        slist.get(-1);
    }

}
```