Programming Language Concepts: Lecture 7

Madhavan Mukund

Chennai Mathematical Institute

madhavan@cmi.ac.in

http://www.cmi.ac.in/~madhavan/courses/pl2009

PLC 2009, Lecture 7, 09 February 2009

Java "generics"

```
public class Node<T> {
  public T data;
  public Node next;
  ...
}
```

I claimed we need to cast the return value to `T` in a generic `LinkedList`

```
public class LinkedList<T>{
  private int size;
  private Node first;

  public T head(){
    T returnval = null;
    if (first != null){
      returnval = first.data;
      first = first.next;
    }
    return (T) returnval; // Cast!!
  }
}

 public void insert(T newdata){
   ...
 }
```

Java "generics"

But this works OK!

```
public class Node<T> {
  public T data;
  public Node<T> next;
  ...
}
```

```
public class LinkedList<T>{
  private int size;
  private Node<T> first;

  public T head(){
    T returnval = null;
    if (first != null){
      returnval = first.data;
      first = first.next;
    }
    return returnval; // No cast!!
  }
}
```

```
 public void insert(T newdata){
   ...
 }
```

Problems with generics

- So we can declare variables of type `Node <T>` inside a generic class with parameter `T`

Problems with generics

- So we can declare variables of type `Node <T>` inside a generic class with parameter `T`
- . . . but we cannot define generic arrays

```
T[] newarray;   // Not allowed!
```

Problems with generics

- So we can declare variables of type `Node <T>` inside a generic class with parameter `T`
- . . . but we cannot define generic arrays

```
T[] newarray;   // Not allowed!
```

- We cannot even have

```
LinkedList<Date>[] newarray;
```

  - Main issue is that arrays are covariant . . .
    - `S` subtype of `T` means `S[]` is compatible with `T[]`

Problems with generics

- So we can declare variables of type `Node <T>` inside a generic class with parameter `T`
- . . . but we cannot define generic arrays

```
T[] newarray;   // Not allowed!
```

- We cannot even have

```
LinkedList<Date>[] newarray;
```

  - Main issue is that arrays are covariant . . .
    - `S` subtype of `T` means `S[]` is compatible with `T[]`
  - . . . while generic types not
    - `S` subtype of `T` does not imply `LinkedList<S>` is compatible with `LinkedList<T>`

Problems with generics

- So we can declare variables of type `Node <T>` inside a generic class with parameter `T`
- . . . but we cannot define generic arrays

```
T[] newarray;   // Not allowed!
```

- We cannot even have

```
LinkedList<Date>[] newarray;
```

  - Main issue is that arrays are covariant . . .
    - `S` subtype of `T` means `S[]` is compatible with `T[]`
  - . . . while generic types not
    - `S` subtype of `T` does not imply `LinkedList<S>` is compatible with `LinkedList<T>`
  - This could create run time type errors

```
ETicket[] elecarr = new ETicket[10];
Ticket[] ticketarr = elecarr;        // OK
ticketarr[5] = new Ticket();         // Not OK!
```

Problems with generics

- The real problem is not . . .

```
T[] newarray;   // Not allowed!
```

Problems with generics

- The real problem is not . . .

```
T[] newarray;   // Not allowed!
```

- . . . but

```
T[] newarray;              // OK
newarray = new T[100]; // Cannot create!
```

Problems with generics

- The real problem is not . . .

```
T[] newarray;   // Not allowed!
```

- . . . but

```
T[] newarray;              // OK
newarray = new T[100]; // Cannot create!
```

- An ugly workaround . . .

```
T[] newarray;
newarray = (T[]) new Object[100];
```

. . . that generates a compiler warning but works!

Exception handling

- Exception — unexpected event that disrupts normal execution

Exception handling

- Exception — unexpected event that disrupts normal execution
- Different levels of severity
  - Divide by zero
  - End of file on read

Exception handling

- Exception — unexpected event that disrupts normal execution
- Different levels of severity
  - Divide by zero
  - End of file on read
- Need to recover from exceptions
  - Take corrective action if possible
  - Abort only if no option left

Exception handling

- Exception — unexpected event that disrupts normal execution
- Different levels of severity
  - Divide by zero
  - End of file on read
- Need to recover from exceptions
  - Take corrective action if possible
  - Abort only if no option left
- Identifying the cause of an exception
  - Need to go beyond rudimentary coding in terms of integer return values, as in C
  - Exceptions have types!

Exception handling in Java

- If an error occurs, an operation throws an exception

Exception handling in Java

- If an error occurs, an operation throws an exception
- Information about the exception is encapsulated as an object

Exception handling in Java

- If an error occurs, an operation throws an exception
- Information about the exception is encapsulated as an object
- Program in which the offending operation occurred should catch the exception object ...

Exception handling in Java

- If an error occurs, an operation throws an exception
- Information about the exception is encapsulated as an object
- Program in which the offending operation occurred should catch the exception object ...
- ... and handle it
  - Analyze the object to determine the cause of the error
  - Take corrective action if possible

Exception handling in Java

- All exceptions are subclasses of `Throwable`
  - Two subclasses, `Error` and `Exception`

Exception handling in Java

- All exceptions are subclasses of `Throwable`
  - Two subclasses, `Error` and `Exception`
  - `Error` — problems beyond program's control

    > *An `Error` . . . indicates serious problems that a reasonable application should not try to catch. Most such errors are abnormal conditions.*

Exception handling in Java

- All exceptions are subclasses of `Throwable`
  - Two subclasses, `Error` and `Exception`
  - `Error` — problems beyond program's control

    > *An `Error` . . . indicates serious problems that a reasonable application should not try to catch. Most such errors are abnormal conditions.*

  - `Exception` — normally caught
    - `RunTimeException` — run time errors reported by JVM: divide-by-zero, array out-of-bounds . . .

Exception handling in Java

`try-catch` structure to handle exceptions

```
try{
      ... // Code that might generate error
   }
catch (ExceptionType1 e){...}
    // Corrective code for ExceptionType1

catch (ExceptionType2 e){...}
   // Corrective code for ExceptionType2
```

Exception handling in Java

`try-catch` structure to handle exceptions

```
try{
      ... // Code that might generate error
   }
catch (ExceptionType1 e){...}
    // Corrective code for ExceptionType1

catch (ExceptionType2 e){...}
   // Corrective code for ExceptionType2
```

- Error in `try` block generates exception object

Exception handling in Java

`try-catch` structure to handle exceptions

```
try{
      ... // Code that might generate error
   }
catch (ExceptionType1 e){...}
    // Corrective code for ExceptionType1

catch (ExceptionType2 e){...}
   // Corrective code for ExceptionType2
```

- Error in `try` block generates exception object
- Exception object is sequentially checked against each `catch`

Exception handling in Java

`try-catch` structure to handle exceptions

```
try{
      ... // Code that might generate error
   }
catch (ExceptionType1 e){...}
    // Corrective code for ExceptionType1

catch (ExceptionType2 e){...}
   // Corrective code for ExceptionType2
```

- Error in `try` block generates exception object
- Exception object is sequentially checked against each `catch`
- What happens if first `catch` is

```
catch (Throwable e1){...}
```

Exception handling in Java

- If some `catch` condition matches, appropriate code is executed
- If no `catch` matches, abort and propagate exception object up one level, to calling class

Exception handling in Java

- May need to do some cleanup (deallocate resources etc)

Exception handling in Java

- May need to do some cleanup (deallocate resources etc)
- This code may be left undone when an exception occurs

Exception handling in Java

- May need to do some cleanup (deallocate resources etc)
- This code may be left undone when an exception occurs
- Add a block labelled `finally`

```
try{
      ...
   }

catch (ExceptionType1 e){...}

catch (ExceptionType2 e){...}

finally{
      ...
  // Always executed, whether try terminates normally
  // or exceptionally.  Use for cleanup statements.
}
```

Customized exceptions

- Don't want negative values in a `LinearList`

Customized exceptions

- Don't want negative values in a `LinearList`
- Define a new class extending `Exception`

```
class NegativeException extends Exception{

  private int error_value;
   // Stores negative value that generated exception

  public NegativeException(String message, int i){
    super(message);  // Appeal to superclass
    error_value = i; // constructor to set message
  }

  public int report_error_value(){
    return error_value;
  }
}
```

Customized exceptions

Inside `LinearList`

```
class LinearList{
  ...
  public add(int i){
    ...
    if (i < 0){
      throw new NegativeException("Negative input",i);
    }
    ...
  }

}
```

Customized exceptions

- A program using `LinearList` should be aware that `LinearList.add()` can result in such an exception

Customized exceptions

- A program using `LinearList` should be aware that `LinearList.add()` can result in such an exception
- Exception should be advertized by `LinearList`

```
class LinearList{
  ...
  public add(int i) throws NegativeException{
   ...
  }
  ...
}
```

Customized exceptions

- A program using `LinearList` should be aware that `LinearList.add()` can result in such an exception
- Exception should be advertized by `LinearList`

```
class LinearList{
  ...
  public add(int i) throws NegativeException{
   ...
  }
  ...
}
```

- Need not advertize exceptions of type `Error` or `RunTimeException`

Customized exceptions

Using `LinearList.add()` with customized exception

```
LinearList l = new LinearList();
...
try{
  ...
  l.add(i);
  ...
}
catch (NegativeException ne){
  System.out.print("Negative input supplied was ");
  System.out.print(ne.report_error_value);
}
...
```

Parameter passing in Java

- Scalars are passed by value
  - No way to write a function to swap two `int`s

Parameter passing in Java

- Scalars are passed by value
  - No way to write a function to swap two `int`s
- Objects are passed by reference

Parameter passing in Java

- Scalars are passed by value
  - No way to write a function to swap two `int`s
- Objects are passed by reference
- How do we swap two objects?

Parameter passing in Java

- Scalars are passed by value
  - No way to write a function to swap two `int`s
- Objects are passed by reference
- How do we swap two objects?

```
class Myclass{
  ...
  public void swap(Myclass p){  // Swap "this" with p
    Myclass tmp;
    tmp = p;
    p = this;
    this = tmp;
  }
}
```

- Will not work!

Parameter passing

- Instead, we must write something like:

```
class Myclass{
  ...
  public void swap(Myclass p){
    Myclass tmp = new Myclass(...);  // Make a new tmp ob
    ... // Copy contents of p into tmp
    ... // Copy contents of this into p
    ... // Copy contents of tmp back into this
  }
}
```

Parameter passing

- Return values?
- Suppose we add a function to `Employee`

```
class Employee{
  ...
  // "accessor" methods
  public Date get_joindate(){ return joindate; }
  ...
}
```

Parameter passing

- Return values?
- Suppose we add a function to `Employee`

```
class Employee{
  ...
  // "accessor" methods
  public Date get_joindate(){ return joindate; }
  ...
}
```

- Now we write

```
Employee e = new Employee(...);
Date d = e.get_joindate();
d.advance(100); // e loses 100 days seniority!
```

Parameter passing

- Return values?
- Suppose we add a function to `Employee`

```
class Employee{
  ...
  // "accessor" methods
  public Date get_joindate(){ return joindate; }
  ...
}
```

- Now we write

```
Employee e = new Employee(...);
Date d = e.get_joindate();
d.advance(100); // e loses 100 days seniority!
```

- Get public access to a private field of `Employee`

Parameter passing

- Return values?
- Suppose we add a function to `Employee`

```
class Employee{
  ...
  // "accessor" methods
  public Date get_joindate(){ return joindate; }
  ...
}
```

- Now we write

```
Employee e = new Employee(...);
Date d = e.get_joindate();
d.advance(100); // e loses 100 days seniority!
```

- Get public access to a private field of `Employee`
- Should make a copy of `joindate` before returning it

Cloning

- `Object` class defines `Object clone(Object o)`

Cloning

- `Object` class defines `Object clone(Object o)`
- Makes a bit-wise copy
  - Nested objects will not be cloned automatically!

Cloning

- `Object` class defines `Object clone(Object o)`
- Makes a bit-wise copy
  - Nested objects will not be cloned automatically!
- To use `clone`, must implement Cloneable

```
class Employee implements Cloneable{
  ...
```

  - Marker interface — empty!

Cloning

- `Object` class defines `Object clone(Object o)`
- Makes a bit-wise copy
  - Nested objects will not be cloned automatically!
- To use `clone`, must implement Cloneable

```
class Employee implements Cloneable{
  ...
```

  - Marker interface — empty!
- Inside `clone()`, expect a check such as

```
Object clone(Object o){
  if (o instanceof Cloneable){
     ... // go ahead and clone
  }else{
     ... // complain and quit
  }
}
```

Packages

- Java has an organizational unit called `package`
  - By default, all classes in a directory belong to a package
  - If neither `public` nor `private` is specified, visibility is with respect to `package`

Packages

- Java has an organizational unit called `package`
  - By default, all classes in a directory belong to a package
  - If neither `public` nor `private` is specified, visibility is with respect to `package`
- Can use `import` to use packages directly

  ```
  import java.math.BigDecimal
  ```

  or

  ```
  import java.math.*
  ```

- All classes in `.../java/math`
- Note that `*` is not recursive

Protected

- `protected` means visible within subtree
- Normally, a subclass cannot change visibility of a function

Protected

- `protected` means visible within subtree
- Normally, a subclass cannot change visibility of a function
- However, `protected` can be made `public`

Protected

- `protected` means visible within subtree
- Normally, a subclass cannot change visibility of a function
- However, `protected` can be made `public`
- `clone()` is defined as `protected`