

Java 5

Prabhu Sunderaraman
prabhu@durasoftindia.com

at

i-flex

December 2004

DuraSoft

<http://www.durasoftindia.com>

Abstract :

The latest version of java development kit 1.5 has brought forward a renovated java language. Java 5, as it is called, is now a factory of new features and loaded enhancements. It looks rich with concepts like boxing, annotations, generics, enumerations and few more. The existing java code may have to undergo a transformation in order to accommodate these new implementations of Java 5. In this presentation we discuss the new features of the language and focus on their importance in realistic development environment.

Targeted Audience :

Software developers, programmers and managers who are involved in day to day technical activities of software development with Java will benefit from this presentation. This session is mainly intended for programmers or people with fairly good programming experience.

Examples :

Each page with a  has an Example.

Download :

Presentation and Code Examples can be downloaded from <http://www.durasoftindia.com>

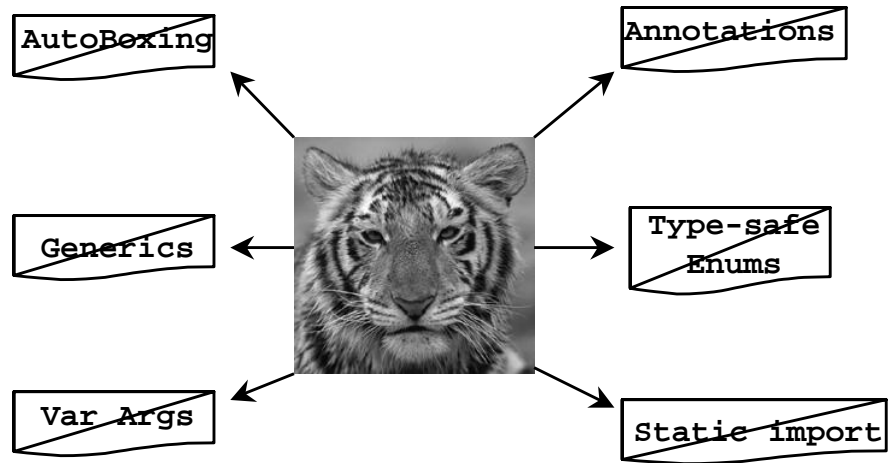
Overview of the Presentation

- Why and What?
- Autoboxing
- Annotations
- Generics
- Enumerations
- Other features
- Language Comparison
- Summary

Why and What?

- Java 2
 - Introduced in 1995
 - Derives its roots from C++, Smalltalk, Objective-C
 - Removes number of pitfalls in C++
- Java 5 or j2se 1.5 or Tiger
 - Derives its roots from C++, .NET!!
 - Transfers programmers job to the compiler
 - More expressive!!!

Taming the Tiger



Overview of the Presentation

- Why and What?
- Autoboxing
- Annotations
- Generics
- Enumerations
- Other features
- Language Comparison
- Summary

Existing approach

- You are working with collection classes
- Create a collection of numbers from 1 to 100
- Maintain a collection of alphabets A-Z
- A function prototype that accepts reference-type
- You want to call the function by passing a primitive-type

- Wrapper Classes!!! manually box and unbox
- Few more key strokes
- Clutters up your code

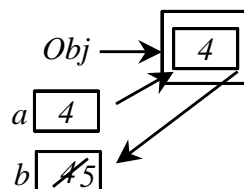
DuraSoft

Java 5-7

AutoBoxing

- Conversion of primitive type to reference type and vice-versa
- Java 5 automates wrapper process
- Used in the collection of primitive-types
- Throws Exception if nulls are unboxed
- Performance overhead involved
- Use it sparingly

```
int a = 4;  
Integer obj = a; // boxing  
int b = obj; // unboxing  
b = 5;
```



DuraSoft

Java 5-8

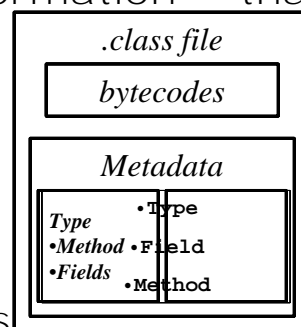


Overview of the Presentation

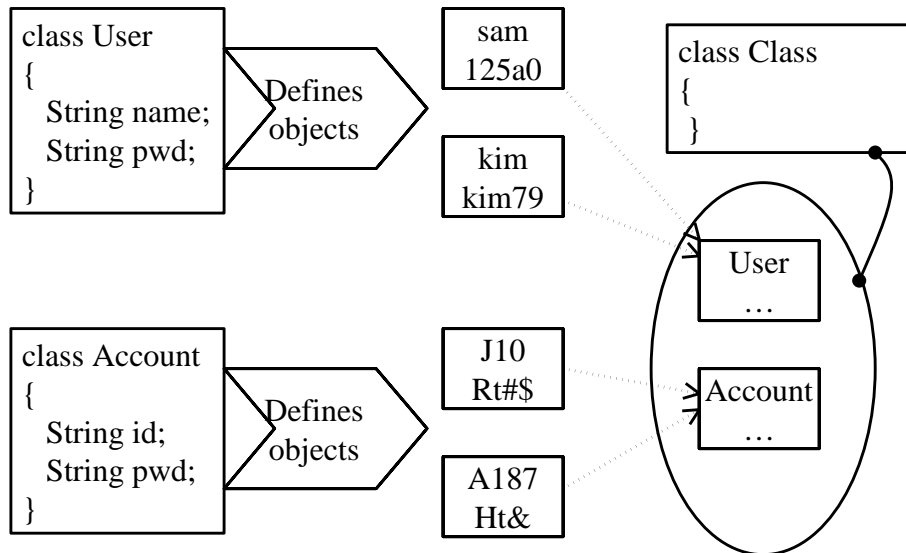
- Why and What?
- Autoboxing
- Annotations
- Generics
- Enumerations
- Other features
- Language Comparison
- Summary

Metadata

- .class file carries information that describes it
- Details about
 - classes
 - methods
 - fields
 - execution and security needs
- Metadata is piggybacked within the class file
- Always stays in synch with the code



objects and metadata



DuraSoft

Reflection mechanism

Java 5-11

Annotations

- Defines characteristics on various subjects
 - classes, methods, properties, etc.
- Similar to the attributes in Microsoft IDL
- Appears preceded by @ before the subject
- **notion of a javadoc tag is generalized**

```
public class Sample  
{  
    @Deprecated  
    public int aField1;  
    public int aField2;  
    @Override public String toString()  
    { return aField2 + "" ;}  
}
```

DuraSoft

Java 5-12



Predefined Annotations

java.lang.Annotation defines

- q Target
- q Retention
- q Inherited
- q Override
- q SuppressWarnings
- q Deprecated

Subjects of an Annotation

- Annotation may appear before the following:
 - Type (Class, interface or enum)
 - Constructor, Method, Field
 - Package, Parameter
- You may write your own Custom annotations
 - specify scope or subject
 - one of above
 - or a combination of above

Writing Custom Annotations

- Interface definition with @ preceding interface keyword
- Method declaration that return primitives
 - No arguments
 - Default values allowed
- Specify possible scope of Annotations (subjects)
- Annotation interfaces automatically extend java.lang.annotation.Annotation.



Writing Custom annotations

```
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.METHOD,
        ElementType.FIELD , ElementType.Type})

@interface AuthorInfo
{
    String name();
    String email();
    String remarks() default "Good!! " ;
}
```

Using Custom Annotations

```
@AuthorInfo(name="prabhu",email="prabhu@durasoftindia.com",
  remarks="Utiliy class")
class Sample
{
  //@AuthorInfo(...)
  //Not allowed here
  public Sample() {}

  @AuthorInfo(name="venkat",email="venkats@agiledeveloper.com")
  public int aField1;

  @AuthorInfo(name="sam",email="sam@yahoo.com")
  public void f1(){ System.out.println("f1" ); }
}
```



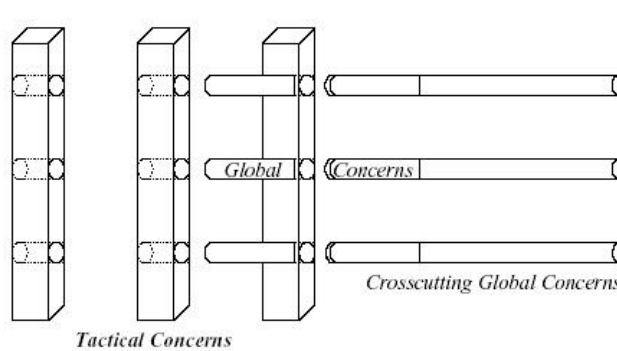
Reflection and Annotation

- Annotations are used to color classes, fields, methods, etc.
- If you place an annotation on a class, how and when is it instantiated?
- Annotations are instantiated only (and each time) when they are requested for



AOP

- Aspect Oriented Programming speaks about with crosscutting concerns that are orthogonal to a system



Annotations aid AOP

EJB 3.0

- Significant changes to the existing versions
- All beans are POJOs
- No deployment descriptors required
- No home interfaces
- Business interfaces only if required
- Changes incorporated using *Annotations*

Annotations in EJB 3.0

- Annotate a simple POJO with predefined annotations

```
@Stateless
@Remote
public class TemperatureProvider
{
    public double getTemperature(String city)
    {
        return Math.random() * 100;
    }
}
Stateless EJB
```



static import

```
import static java.lang.Math.*;
```

- Analogous to normal import
- Imports static members from classes
- Unqualified access to static members
- Useful for frequent access to static members
- Use it sparingly!!
- May lead to maintainability problems

Overview of the Presentation

- Why and What?
- Autoboxing
- Annotations
- Generics
- Enumerations
- Other features
- Language Comparison
- Summary

for-each construct

- Useful to execute for each element in a collection
 - *arr is a string array*

```
for(String s : arr)
{
    // s refers to an element in array arr, first, second,...
    System.out.println(s);
}
```
- Combined with generics to iterate over a collection
- *Used for retrieval purposes only*

Existing Approach

- Type-of the Object in a Collection is known at compile-time
- Cast required to retrieve objects
- Prone to throw ClassCastException
- Bit annoying writing type-cast code
- Enforce type-safety without type-cast during compile-time and not introduce clutter ?

Generics

- Abstraction over types
- Collections can communicate the type of the elements
- Avoid run-time cast exceptions
- Type-cast is implicit
- Resembles Templates in C++
- Compile-time gimmick, no VM support

```
Vector<Movie> movies = new Vector<Movie>();  
movies.add(new Action());  
movies.add(new Comedy());  
// compile-time error movies.add("Romance");
```



Wildcards and inheritance

- Used when no specific information about the type passed is known at compile-time


```
public void playAll(List<?> movies)    {}  
public void playAll(List<? extends Movie> movies)  {}
```

- Interfaces are represented in a similar fashion

```
public void playAll(List<? extends IMovie> movies)    {}
```

*"extends" used instead of
"implements"!! Misleading syntax*

DuraSoft

Java 5-27 

Generic Methods and Types

Provides Generalized Implementation

```
public class DVDCollection<T>  
{  
    Vector<T> movies = new Vector<T>();  
    ... ..  
}
```

```
public static <T extends Movie> void play(T t)  
{  
    t.play();  
}  
  
play(new Action());  
play(new Comedy());
```

← What's the point?

DuraSoft

Java 5-28 

Erasure

- In Generics
 - “type of anything” is actually “type of Object”
- Compiler *erases* generic code to non-generic code

```
public void play(T movie)
{
    // compile-time error
    //movie.play();
}
```

- Narrows down the usage of generics
- Erasure is provided for compatibility

Overview of the Presentation

- Why and What?
- Autoboxing
- Annotations
- Generics
- Enumerations
- Other features
- Language Comparison
- Summary

Existing Approach

- Representation of an enumerated list of items
- Values for the items may be a constant
- Constant interfaces provide a solution
 - Suffers from unsafe type !!!
- Implementation of enum-pattern using classes
 - Tedious

Enumerations

- Java 5 provides Enums
- Not the usual int enums
- Ordering is not an issue
- Ensures type-safety during compile-time
- More powerful and flexible

```
public enum Food
{
    Breakfast, Lunch, Dinner;
    public String toString()
    {
        // custom
    }
}
```



java.lang.enum

- `compareTo(enumType)`
 - Compares for the order
- `name()`
 - Name of the enum constant
- `ordinal()`
 - Position of the enum constant
- `valueOf(enumType,name)`
 - Enum constant of the specified enum type
- `values()`
 - collection of the values present in the list

Enum semantics

- Enums in java 5 are special class declarations
- Can have fields, methods and constructors
- Used in Collection classes with generics
- switch-case statements can use enum types

Overview of the Presentation

- Why and What?
- Autoboxing
- Annotations
- Generics
- Enumerations
- Other features
- Language Comparison
- Summary

Passing variable # of arguments

- Like ellipsis (...) in C++
 - except for enhanced type safety
- Allows passing several arguments of certain type
- Three periods after the type of the argument
- Only one allowed per method
- Must be the last one in the parameter list

```
public void f(int i,double ... d)
{
    for(double d1:d)
        System.out.println(d1);
}

//call
f(1,2,34,6,7,8);
f(1,new double[]{1,2,3});
```



PrintStream

- *printf*(String format, Object... args)
- Formatted output as in C
- Variable argument list used
- AutoBoxing is handy

```
System.out.printf( "%4d--%s-(%h)%n",  
                   12,"Seminar!",new X()  
                   );
```

java.util.Scanner

- Reading input from command line is cumbersome
- Scanner class provides quick Console I/O operations
- Works like StreamTokenizer
- Methods available on a Scanner Object
 - next(), nextInt(), nextBoolean(), nextDouble(), etc.,

EnumSet and EnumMap

- Specialized classes to add Enum type values
- More compact storage and efficient retrieval
- Unsynchronized collections
- EnumSet
 - Elements belong to a single enum type
- EnumMap
 - Key type come from single enum type

Underplayed Feature

- Covariant return types
- Defies method overriding concepts
- Very less focus on this addition
- boon or bane?

```
public class A {  
    public Object foo()    { ... }  
}  
class B extends A {  
    public String foo()    { ... }  
}
```

Overview of the Presentation

- Why and What?
- Autoboxing
- Annotations
- Generics
- Enumerations
- Other features
- Language Comparison
- Summary

Java Vs .NET Vs C++

Feature	C++	.NET
Annotations	--	A
Generics	A (Templates)	A (in 2.0)
Type-safe Enums	A(int-based)	A(int-based)
Variable Arguments	A	A
Static import	--	--
AutoBoxing	--	A

Attributes

- Equivalent of Annotations in .NET
- More effective and transparent
- Custom Attributes are implemented as classes, not as interfaces
- **Attributes are always retained during runtime**

```
[Serializable]
public class Sample
{
    private int aField1,aField2;
    [System.Obsolete("Use inc instead")]
    public void increment(){aField1++;}
    public void inc() {aField2++;}
}
```

Templates

- Templates in C++ serve as roots for Generics in Java
- Generics in .NET 2.0 have a better model
- Templates provide latent typing, a very useful feature in C++

```
template<class T> void foo(T movie)
{
    movie.play();
}
```

Variable arguments / Boxing

- Ellipsis in C has been around for years
 - Pretty cumbersome
- Java provides variable arguments exactly the .NET way
- AutoBoxing feature similar to .NET Boxing/Unboxing

Overview of the Presentation

- Why and What?
- Autoboxing
- Annotations
- Generics
- Enumerations
- Other features
- Language Comparison
- Summary

Summary

- Has some learning curve as the syntax is slightly confusing
- Need to know the limits of Generics before using them
- Requires a bit more discipline in developing the classes using Annotations
- Let's crack some code and have fun!!

References

- <http://java.sun.com>
- <http://www.javaworld.com>