

# Concepts of Object-Oriented Programming

**Prof. Dr. Peter Müller**

Chair of Programming Methodology

Exercises 2: Java Overview



Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich

## 2. Java Overview

- Quick overview of Java
- Focus on main elements for object-oriented programming
- Look at trickier problems



Java, J2EE, J2SE, J2ME, and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

# How to compile and run Java programs

- Java source code is saved in `.java` files
- Every public class must be put into its own `.java` file
- `javac` is used to compile Java files:

```
javac Test.java
```

- Compilation results in one or more `.class` files
- Set correct `CLASSPATH` environment variable
- One of your classes needs a special method:

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

- To execute the main method of class `Test` type:

```
java Test
```

# Classes and Objects – Basics

- Classes are the basic building block of Java
- Every attribute and method belongs to a class
- Every class inherits from the base class  
`java.lang.Object`
- `Object` is the root of all classes, even if it is not mentioned as superclass
- `Object` provides basic features like string conversion, comparison, cloning, synchronization and meta information

# Why a class `Object` as root of the type tree?

- Create an array that can contain any object:

```
Object[] arr = new Object[10];  
arr[0] = "A String here...";  
arr[1] = new Thread();
```

- Methods that can take any object as argument:

```
public void log( Object message ) {  
    System.err.println("Message: " + message);  
}
```

- Ensure some standard methods, e.g. `toString` for String conversion and `equals` for equality

# Problem with Arrays

```
void readArray( Object[] arr ) {  
    System.out.println("Array[0]=" + arr[0]);  
}
```

```
void writeArray( Object[] arr ) {  
    arr[0] = "My array!"; // move other elems ...  
} java.lang.ArrayStoreException!
```

```
Integer[] ia = new Integer[2];  
ia[0] = new Integer(5);  
ia[1] = new Integer(6);
```

```
readArray( ia );  
writeArray( ia );
```

# Primitive Data Types

- **These are:** `boolean`, `byte`, `short`, `int`, `long`, `char`, `float`, `double`
- They are not subclasses of `Object` and can not be used where objects are required (e.g. Collections)
- In those cases the wrapper classes can be used, e.g. `Integer` or `Float`
- Autoboxing added in Java 1.5
- Precision and behavior of all types exactly defined
- ```
Object[] arr = new Object[10];  
arr[0] = 10;  
arr[1] = new Integer(10);
```

**Compilation Error!** Java 1.4  
**Ok.**

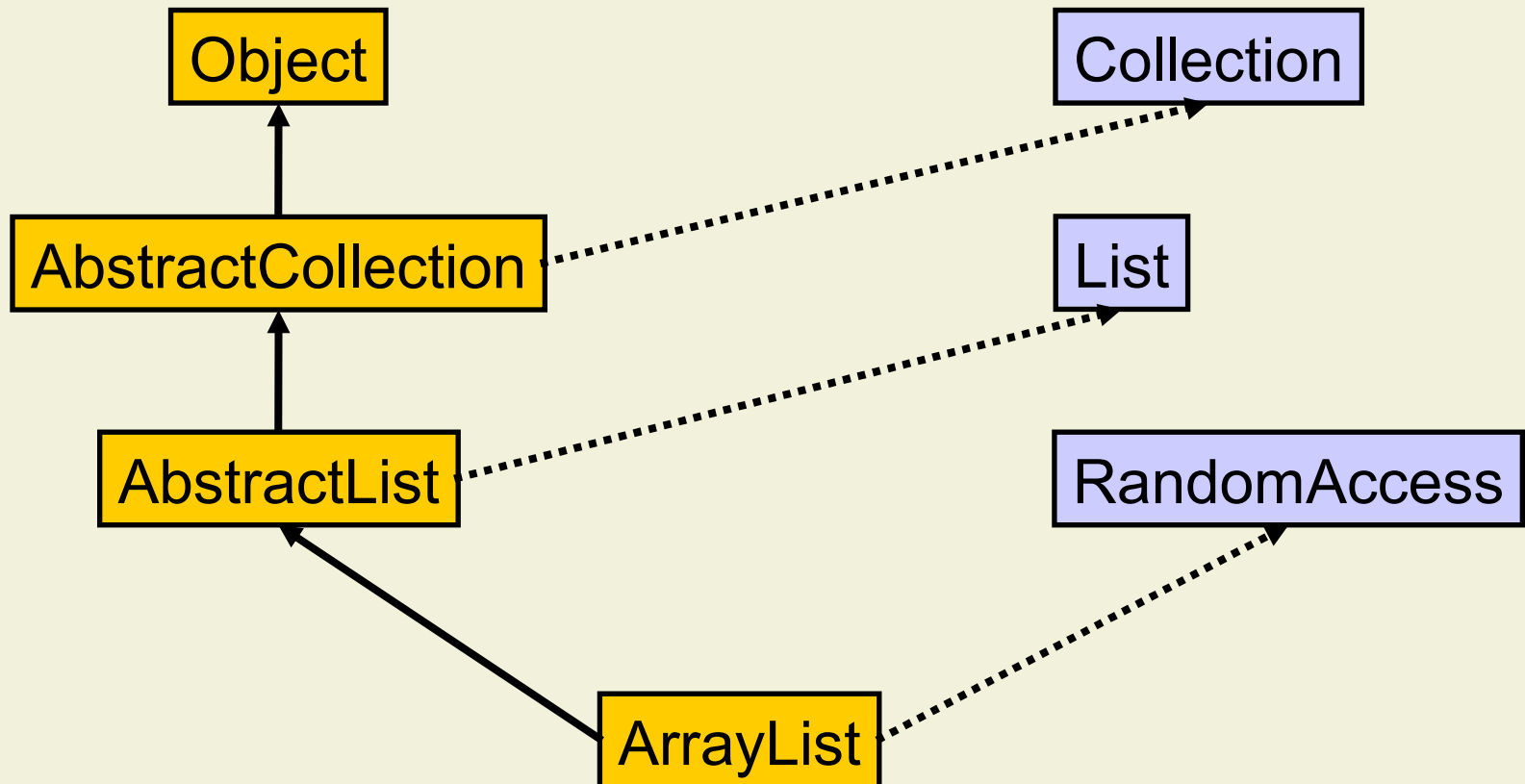
# Classes and Objects – Inheritance

- Single Implementation Inheritance:
  - Only one other class can be extended
  - Methods and attributes of the superclass are inherited and can also be used on the subclass (if visible)
  - Subtyping and Inheritance at the same time  
→ Subclassing
  
- Multiple Interface Subtyping:
  - Interfaces just specify which methods a class must have to comply with the interface
  - No implementation associated with the interfaces



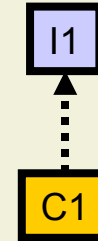
# Inheritance Example

- Class `java.util.ArrayList`:



# Quiz about Type Casts 1:

```
interface I1 {}  
class C1 implements I1 {}
```



```
public class Test1 {  
    public static void main( String[] args ) {  
        C1 c1 = new C1();  
        I1 i1 = (I1) c1;  
    }  
}
```

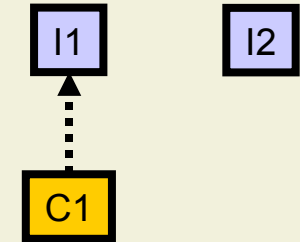
- Compile-time errors?
- Run-time errors?

No.

No.

## Quiz about Type Casts 2:

```
interface I1 {}  
interface I2 {}  
class C1 implements I1 {}
```



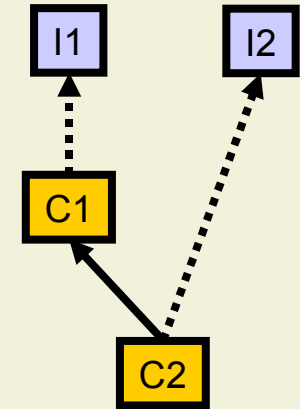
```
public class Test2 {  
    public static void main( String[] args ) {  
        C1 c1 = new C1();  
        I2 i2 = (I2) c1;  
    }  
}
```

- Compile-time errors? No.
- Run-time errors? **java.lang.ClassCastException!**

## Quiz about Type Casts 3:

```
interface I1 {}
interface I2 {}
class C1 implements I1 {}
class C2 extends C1 implements I2 {}
```

```
public class Test3 {
    public static void main( String[] args ) {
        C1 c1 = new C2();
        I2 i2 = (I2) c1;
    }
}
```



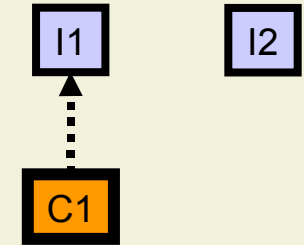
- Compile-time errors?
- Run-time errors?

No.

No.

## Exercise 2.9

```
interface I1 {}  
interface I2 {}  
final class C1 implements I1 {}
```



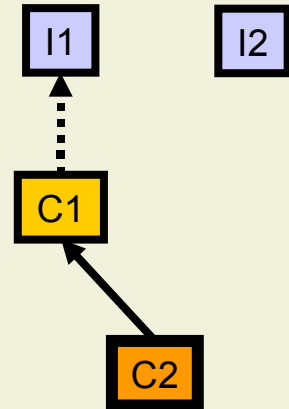
```
public class Ex1 {  
    public static void main( String[] args ) {  
        C1 c1 = new C1();  
        I2 i2 = (I2) c1;  
    }  
}
```

- **Compilation Error: incompatible types**

## Exercise 2.10

```
interface I1 {}  
interface I2 {}  
class C1 implements I1 {}  
final class C2 extends C1 {}
```

```
public class Ex2 {  
    public static void main( String[] args ) {  
        C1 c1 = new C2();  
        I2 i2 = (I2) c1;  
    }  
}
```



- Runtime Error: `java.lang.ClassCastException`

## Exercise 2.11

```
class C1 {  
    int age = 99;  
    int getAge() {  
        return age; }  
}  
  
class C2 extends C1 {  
    int age = 22;  
    int getAge() {  
        return age; }  
}
```

### ■ Output:

```
c1.age:                99  
c1.getAge():           99  
c2.age:                22  
c2.getAge():           22  
c1=c2.age:             99  
c1=c2.getAge():        22
```

# Overloading and Overriding

- The same method name can be used with different parameter types → Overloading
- The most specific method signature is selected for each method invocation
- Subclasses can define a different implementation of a method → Overriding
- For overriding all method parameter types have to stay the same, otherwise the method is overloaded



# Overloading Example

```
static void func( String str, Object obj ) {  
    System.out.println("String, Object");  
}
```

```
static void func( Object obj, String str ) {  
    System.out.println("Object, String");  
}
```

```
public static void main( String[] args ) {  
    func( "Name", new Integer(10) );  
    func( new Integer(10), "Name" );  
    func( "Name", "10" );  
}
```

Compilation Error!

# Exception Handling

## Basic syntax:

```
try {  
    // dangerous code or ...  
    throw new MyException();  
    ...  
} catch( MyException me ) {  
    // handle my own exception  
} catch( Exception e ) {  
    // handle other exceptions  
} finally {  
    // do something that definitely has to happen  
}
```

# Exception Handling

- Catch-Blocks contain the code that handles the exceptional situation; the first catch-block whose type is a supertype of the current exception will be used → order the exceptions from most specific to least specific
- Finally-Block contains code that needs to be executed after the try-block, with or without an exception occurring
- Exceptions that are not caught need to be declared in the method signature! Example:  
**void** m() **throws** MyException { ... }

# Assertions (since Java 1.4)

- Two versions of assertions:

```
assert Expression1 ;
```

```
assert Expression1 : Expression2 ;
```

- Java evaluates `Expression1` and if it is false throws an `AssertionError` with the value of `Expression2` as detail message, if present.

- Compile like this:

```
javac -source 1.4 MyClass.java
```

- At runtime enable/disable assertions selectively per package or class:

```
java -enableassertions:pack.age... Test
```

- <http://java.sun.com/j2se/1.4.2/docs/guide/lang/assert.html>

# Since Java 1.5 – Generics

- Defining a generic class:

```
public interface List<E> {  
    void add(E x);  
    Iterator<E> iterator();  
}
```

- Use:

```
List<String> words =  
    new ArrayList<String>();
```

# Since Java 1.5 – Generics

- Generic Methods:

```
public <T extends E> boolean  
addAll(Collection<T> c);
```

- Wildcards:

```
void printCollection(Collection<?> c) ...
```

- Can get tricky:

```
public static <T extends C<? super T>>  
T max(Collection<T> coll)
```

- <http://java.sun.com/j2se/1.5/pdf/generics-tutorial.pdf>

# Generics and Overloading

```
public class Test15<T> {  
    public void m( T t ) { ... }  
    public void m( String s ) { ... }  
    public static void main( String[] args ) {  
        Test15<Object> to = new Test15<Object>();  
        to.m( (Object) "Hello World!" );  
        to.m( "Hello World!" );  
  
        Test15<String> ts = new Test15<String>();  
        ts.m( "Hello World!" );  
    }  
}
```

Generic m  
String m  
Ambiguity!

## Since Java 1.5

- Annotations, e.g. `@Deprecated` and `@Override` :

`@Override`

```
public void foo (Param p) { }
```

- Enhanced `for`-loop:

```
void cancelAll (Collection c) {  
    for (Object o : c)  
        ((TimerTask) o).cancel();  
}
```



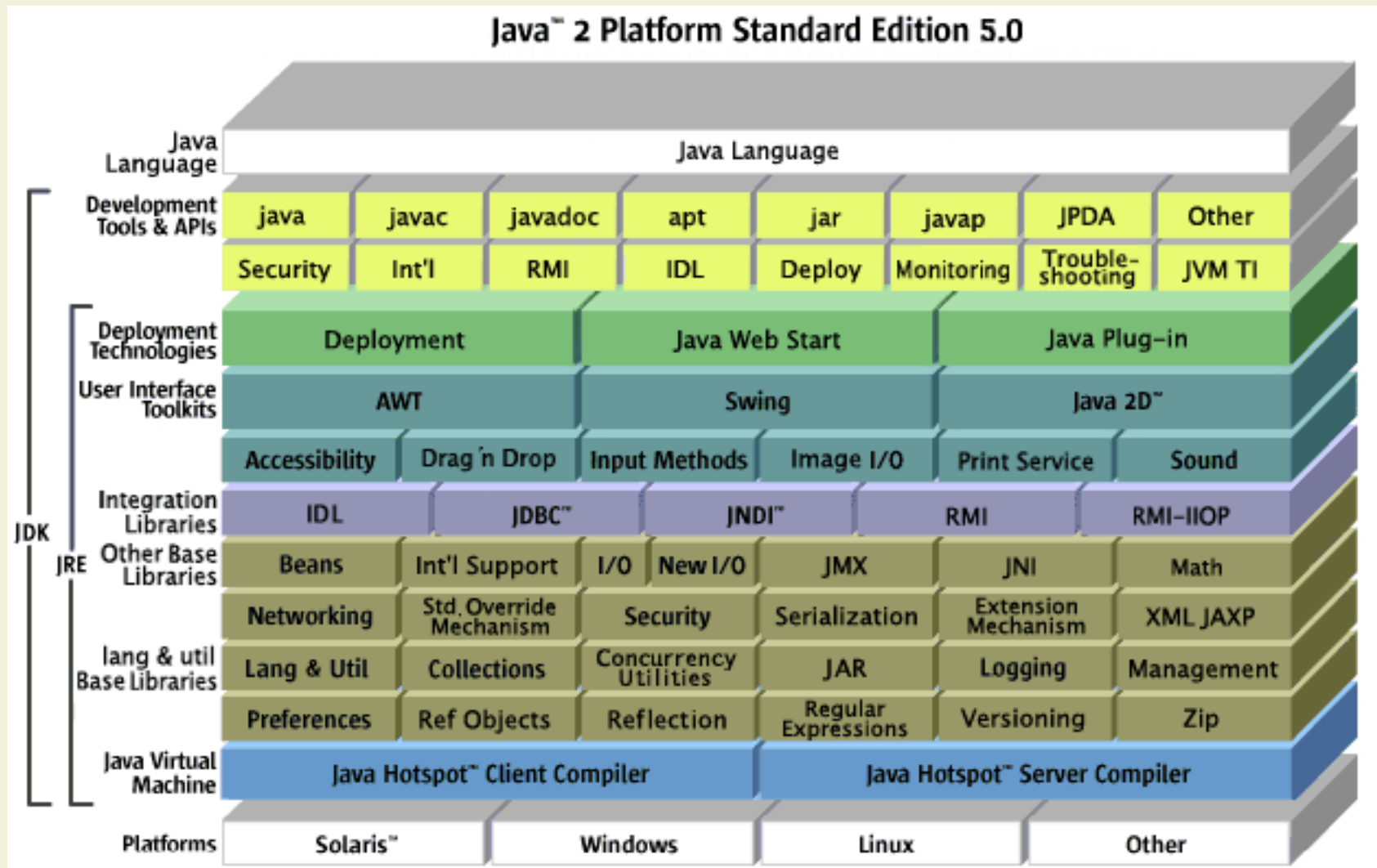
# Since Java 1.5

- Autoboxing/unboxing

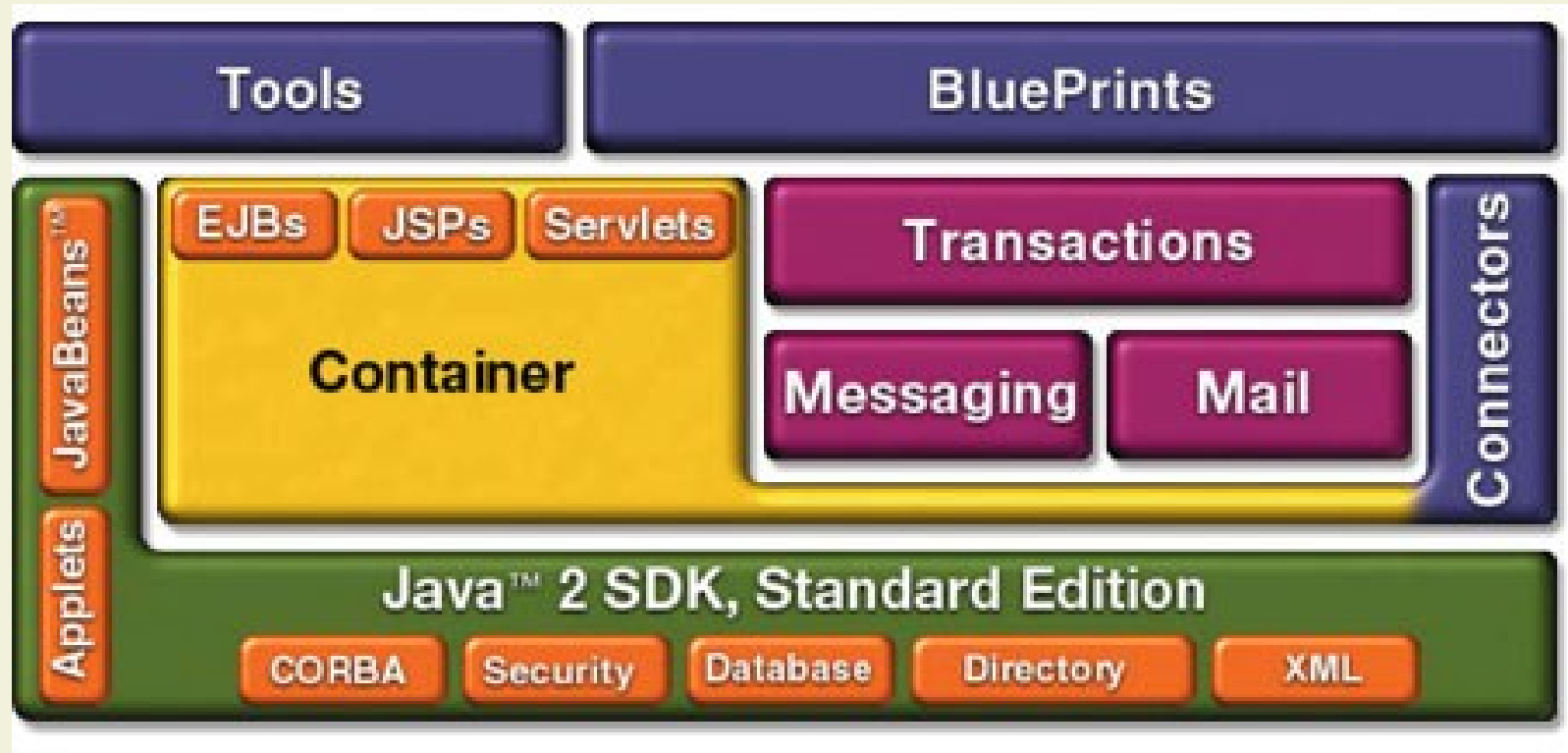
```
public class Freq {  
    public static void main(String args[]) {  
        Map<String, Integer> m =  
            new TreeMap<String, Integer>();  
        for (String word : args)  
            m.put(word, m.get(word) + 1);  
        System.out.println(m);  
    }  
}
```

- Other things: typesafe enums, static import, metadata, ...

# Java 2 Platform, Standard Edition



# Java 2 Platform, Enterprise Edition



# Other Java Technologies

- Java Access Bridge
- Java Advanced Imaging
- Java Authentication and Authorization Service (JAAS)
- Java Communications API (JCA)
- Java Cryptography Extension (JCE)
- Java Data Objects (JDO)
- JavaMail API
- Java Management Extensions (JMX)
- Java Media Framework (JMF)
- Java Naming and Directory Interface (JNDI)
- Java Secure Socket Extension (JSSE)
- Java Speech API
- Java 3D API

# Questions?