Java_Advanced

Advanced Concepts

- Object Orientation
- Data Structures
- Multithreaded programming
- Serialization mechanisms
- Generics
- Collections Framework

OO Java Concepts

- Inheritance
- Polymorphism
- Abstraction
- Encapsulation
- Overriding

OO-Inheritance

- IS-A Relationship
- HAS-A Relationship
- Extending a **Class**
- Implementing an **Interface**

Polymorphism

- The ability of an object reference to be used as if it referred to an object with different forms

  class Rectangle extends Polygon implements Comparable

- An object whose *dynamic type* is Rectangle can behave as all of the following types: Rectangle, Polygon, Comparable, Object.

Abstraction

- A simplified representation of something that is potentially quite complex
- Object-oriented design often involves finding the right level of abstraction at which to work when modeling real-life objects
- If the level is too high, then not enough detail will be captured. If the level is too low, then a program could be more complex and difficult to create and understand than it needs to be

Encapsulation

Safeguarding the state of an objects by defining its attributes as private and channeling access to them through accessor and mutator methods.

Method Overriding

- A method defined in a super class may be overridden by a method of the same name defined in a sub class
- The two methods must have the same name and number and types of formal arguments
- Any checked exception thrown by the sub class version must match the type of one thrown by the super class version, or be a sub class of such an exception
- Check: *overriding for breadth*, *overriding for chaining and overriding for restriction*

Method Overriding - Rules

- The argument list should be exactly the same as that of the overridden method.
- The return type should be the same or a subtype of the return type declared in the original overridden method in the superclass.
- The access level cannot be more restrictive than the overridden method's access level.
- Instance methods can be overridden only if they are inherited by the subclass.
- A method declared final cannot be overridden.
- A method declared static cannot be overridden but can be re-declared.
- If a method cannot be inherited, then it cannot be overridden.
- A subclass within the same package as the instance's superclass can override any superclass method that is not declared private or final.
- A subclass in a different package can only override the non-final methods declared public or protected.
- Constructors cannot be overridden

Method Overloading

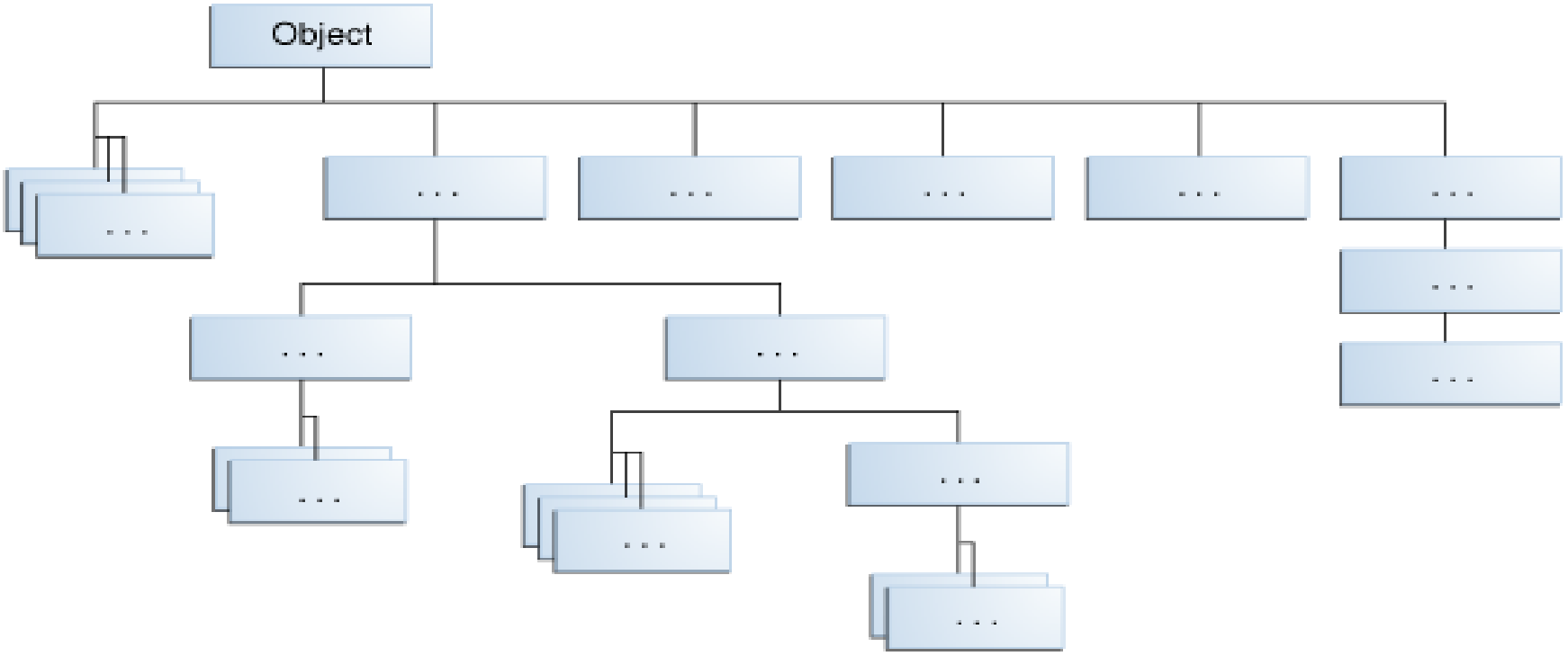
1. Change the number of method parameters
2. Change the parameter types

Συχνά λάθη:

1. Overload return type
2. Change parameter name

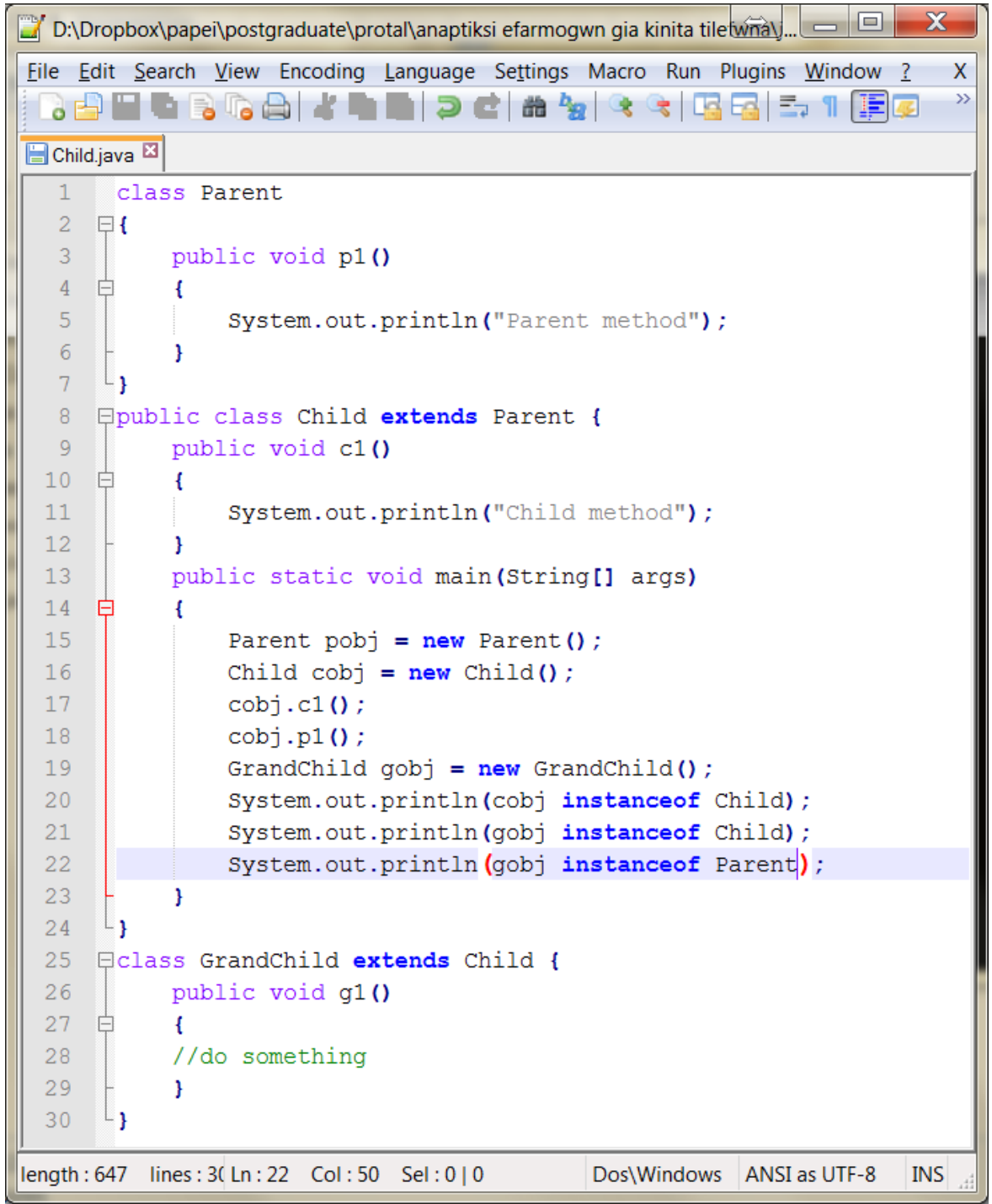
Method Header

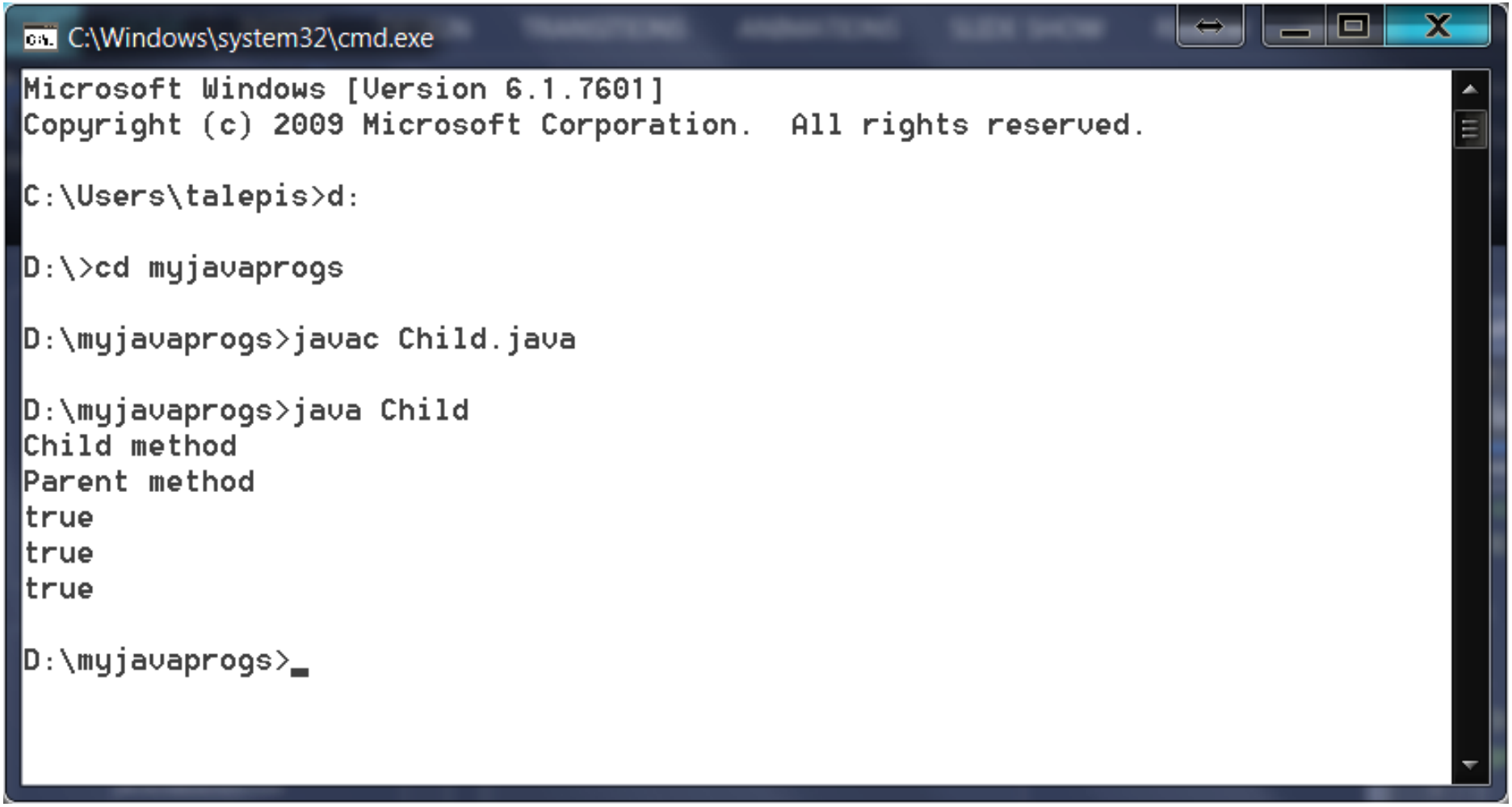
- The header of a method, consisting of the method name, its result type, formal arguments and any exceptions thrown
- Also known as a *method signature*


Object
. . .
. . .
. . .
. . .
. . .
. . .
. . .
. . .
. . .
. . .
. . .
. . .
. . .
. . .
. . .

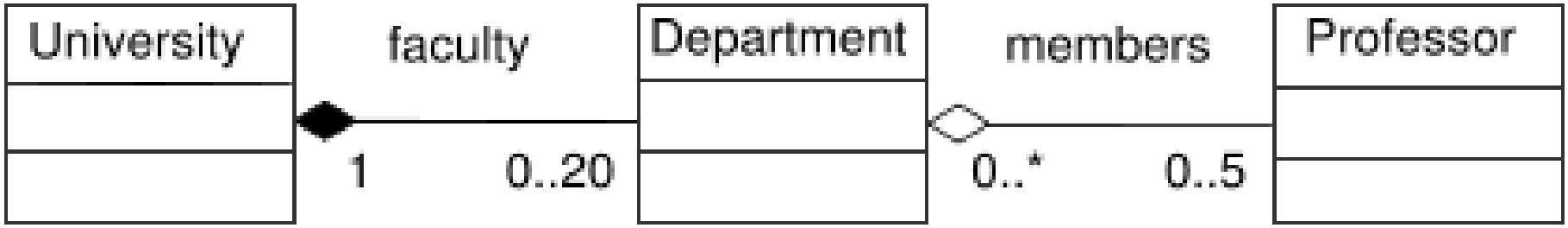
Inheritance example

```
public class Animal{
}

public class Mammal extends Animal{
}

public class Coldblooded extends Animal{
}

public class Egglaying extends Animal{
}

public class Cat extends Mammal{
}
```

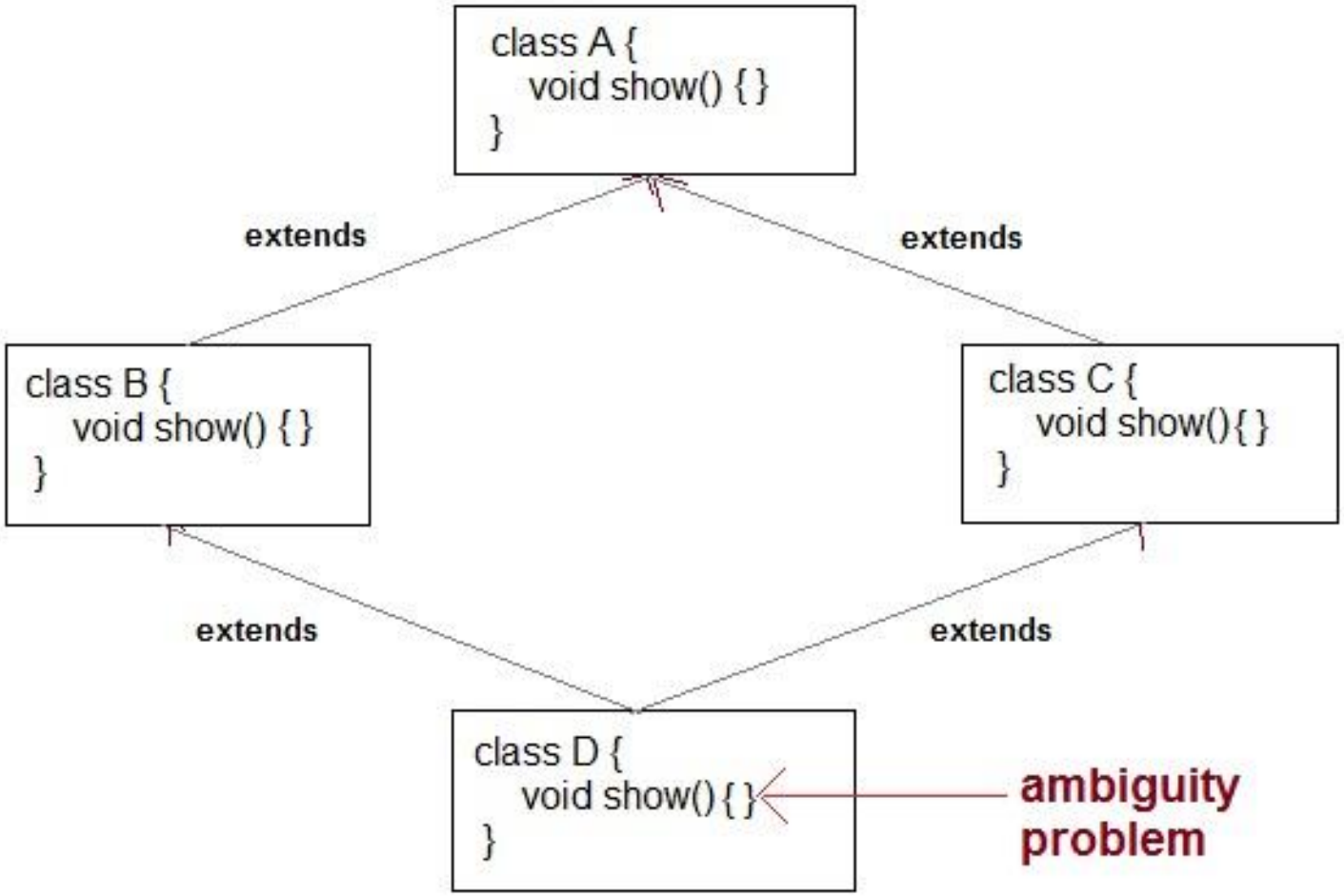

D:\Dropbox\papei\postgraduate\protal\anaptiksi efarmogwn gia kinita tilefwna\...
File Edit Search View Encoding Language Settings Macro Run Plugins Window ? X
Child.java
1 class Parent
2 {
3 public void p1()
4 {
5 System.out.println("Parent method");
6 }
7 }
8 public class Child extends Parent {
9 public void c1()
10 {
11 System.out.println("Child method");
12 }
13 public static void main(String[] args)
14 {
15 Parent pobj = new Parent();
16 Child cobj = new Child();
17 cobj.c1();
18 cobj.p1();
19 GrandChild gobj = new GrandChild();
20 System.out.println(cobj instanceof Child);
21 System.out.println(gobj instanceof Child);
22 System.out.println(gobj instanceof Parent);
23 }
24 }
25 class GrandChild extends Child {
26 public void g1()
27 {
28 //do something
29 }
30 }
length : 647 lines : 3(Ln : 22 Col : 50 Sel : 0 | 0 Dos\Windows ANSI as UTF-8 INS


C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
C:\Users\talepis>d:
D:\>cd myjavaprogs
D:\myjavaprogs>javac Child.java
D:\myjavaprogs>java Child
Child method
Parent method
true
true
true
D:\myjavaprogs>

Composition-Aggregation



Multiple Inheritance? **No**
Diamond Problem



Basic Data Stuctures

- Enumeration
- BitSet
- Vector
- Stack
- Dictionary
- Hashtable
- Properties

Collection Classes

1 AbstractCollection

Implements most of the Collection interface.

2 AbstractList

Extends AbstractCollection and implements most of the List interface.

3 AbstractSequentialList

Extends AbstractList for use by a collection that uses sequential rather than random access of its elements.

4 LinkedList

Implements a linked list by extending AbstractSequentialList.

5 ArrayList

Implements a dynamic array by extending AbstractList.

6 AbstractSet

Extends AbstractCollection and implements most of the Set interface.

7 HashSet

Extends AbstractSet for use with a hash table.

8 LinkedHashSet

Extends HashSet to allow insertion-order iterations.

9 TreeSet

Implements a set stored in a tree. Extends AbstractSet.

10 AbstractMap

Implements most of the Map interface.

11 HashMap

Extends AbstractMap to use a hash table.

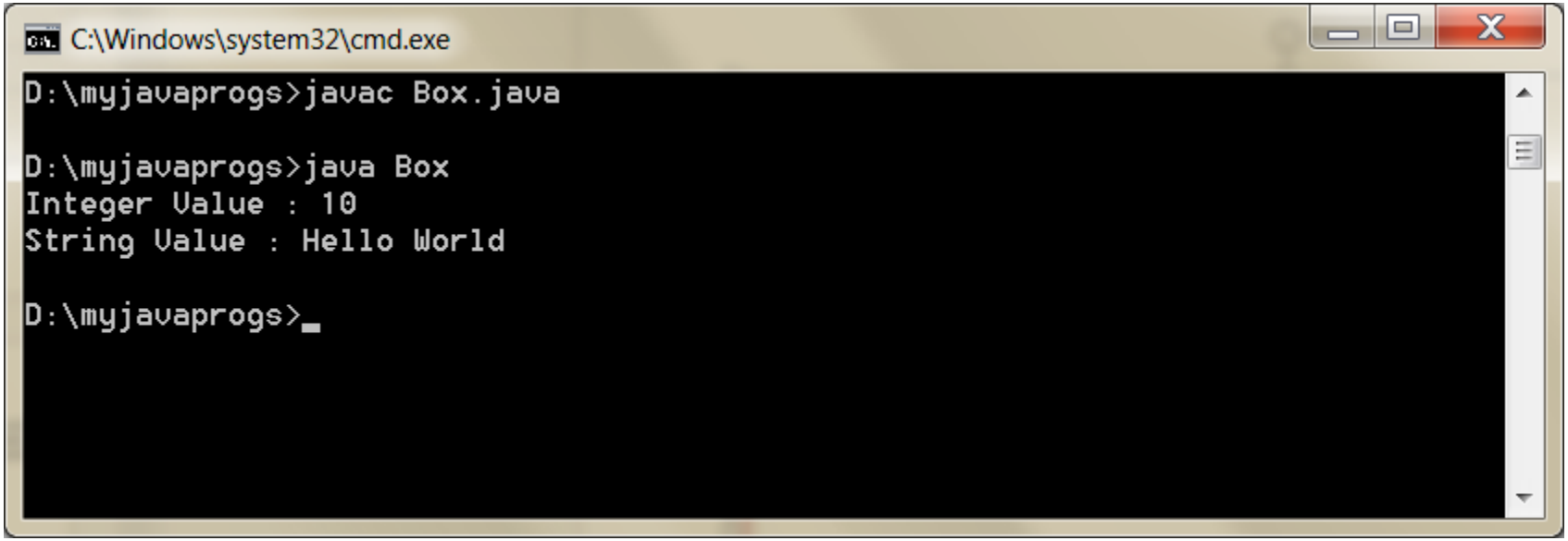
Generics

Generics enable types (classes and interfaces) to be parameters when defining classes, interfaces and methods

Generic Methods and Classes

- Java **Generic** methods and generic classes enable programmers to specify, with a single method declaration, a set of related methods or, with a single class declaration, a set of related types, respectively
- Using Java Generics we might write a generic method for sorting an array of objects, then invoke the generic method with Integer arrays, Double arrays, String arrays and so on, to sort the array elements

```java
public class Box<T> {

  private T t;

  public void set(T t) {
    this.t = t;
  }

  public T get() {
    return t;
  }

  public static void main(String[] args) {
     Box<Integer> integerBox = new Box<Integer>();
     Box<String> stringBox = new Box<String>();

     integerBox.set(new Integer(10));
     stringBox.set(new String("Hello World"));

     System.out.println("Integer Value : " +integerBox.get());
     System.out.println("String Value : " +stringBox.get());
  }
}
```


C:\Windows\system32\cmd.exe
D:\myjavaprogs>javac Box.java
D:\myjavaprogs>java Box
Integer Value : 10
String Value : Hello World
D:\myjavaprogs>

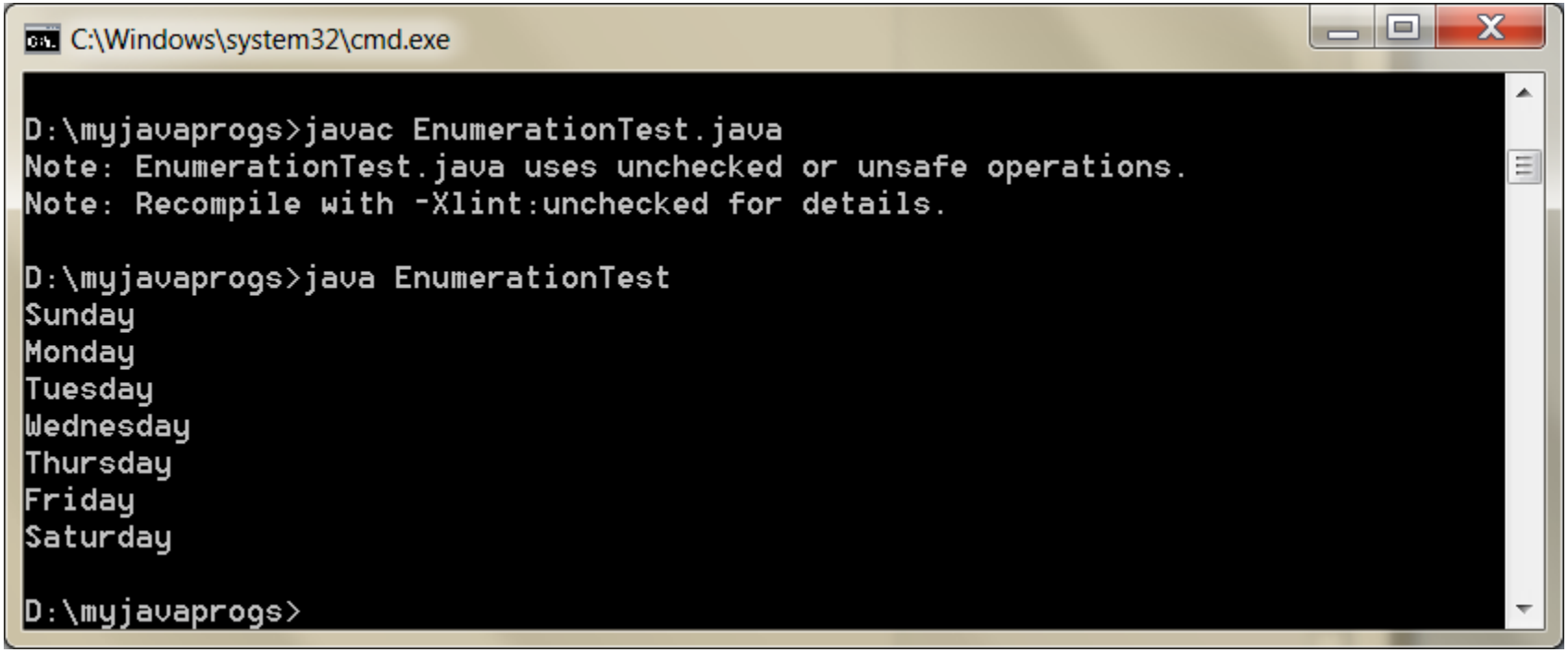
Java - Serialization

- An object represented as a sequence of bytes that includes the object's data as well as information about the object's type and the types of data stored in the object
- Object serialization to a file (*.ser)
- Object deserialization from a file (object recreated in memory)
- Platform independence!!

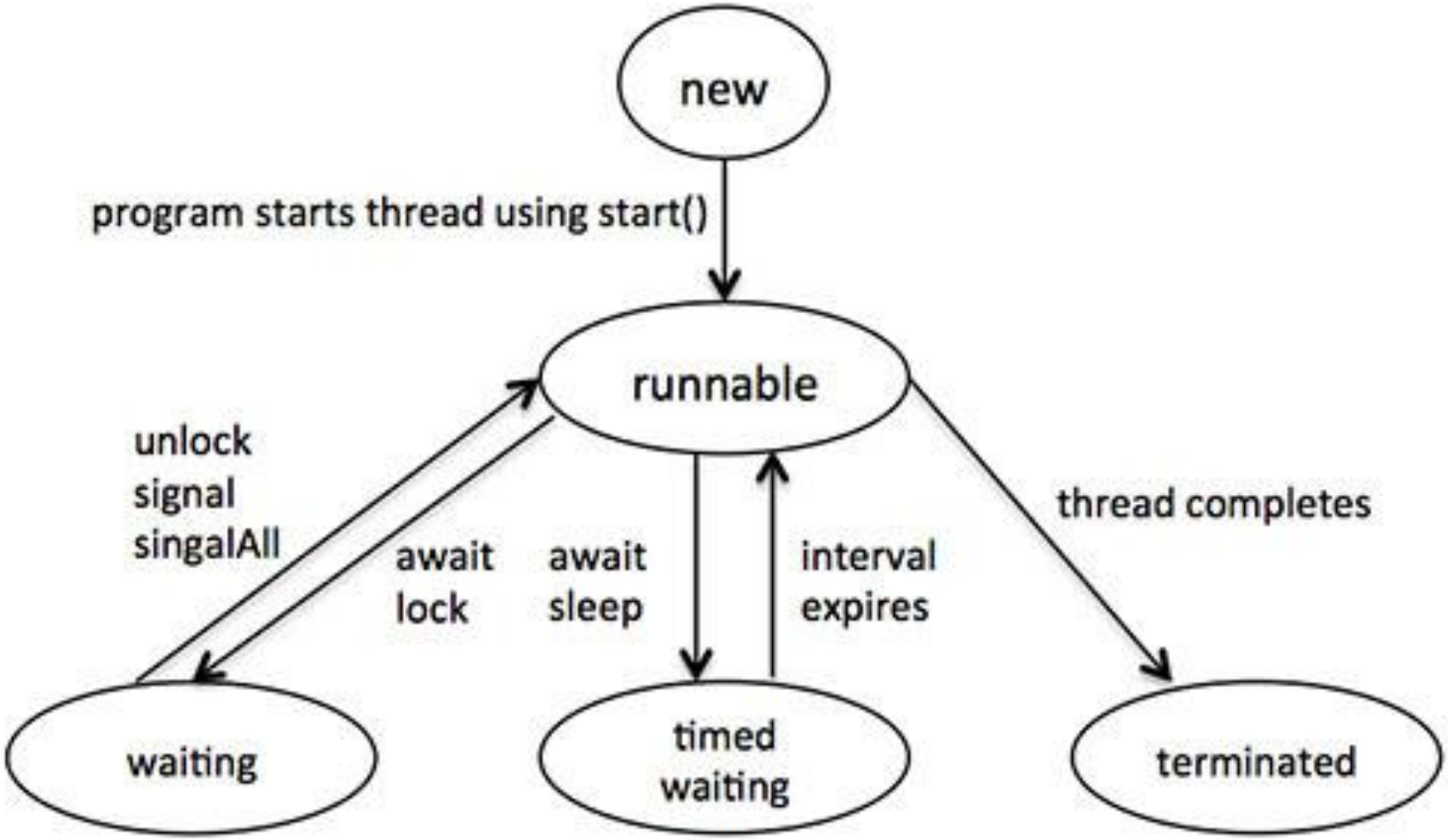
Enumeration Interface

The Enumeration interface defines the methods by which you can enumerate the elements in a collection of objects

```java
import java.util.Vector;
import java.util.Enumeration;

public class EnumerationTest {

   public static void main(String args[]) {
      Enumeration days;
      Vector dayNames = new Vector();
      dayNames.add("Sunday");
      dayNames.add("Monday");
      dayNames.add("Tuesday");
      dayNames.add("Wednesday");
      dayNames.add("Thursday");
      dayNames.add("Friday");
      dayNames.add("Saturday");
      days = dayNames.elements();
      while (days.hasMoreElements()){
         System.out.println(days.nextElement());
      }
   }
}
```


C:\Windows\system32\cmd.exe
D:\myjavaprogs>javac EnumerationTest.java
Note: EnumerationTest.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
D:\myjavaprogs>java EnumerationTest
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
D:\myjavaprogs>

Multithreading – Thread Life Cycle



Java Threads

- Extend Thread Class
- Thread Priorities
  - **(MIN_PRIORITY<NORMAL_PRIORITY<MAX_PRIORITY)**
- Thread Synchronization
- Interthread Communication
- Thread Deadlock