Lecture 6
summary of Java SE

presentation

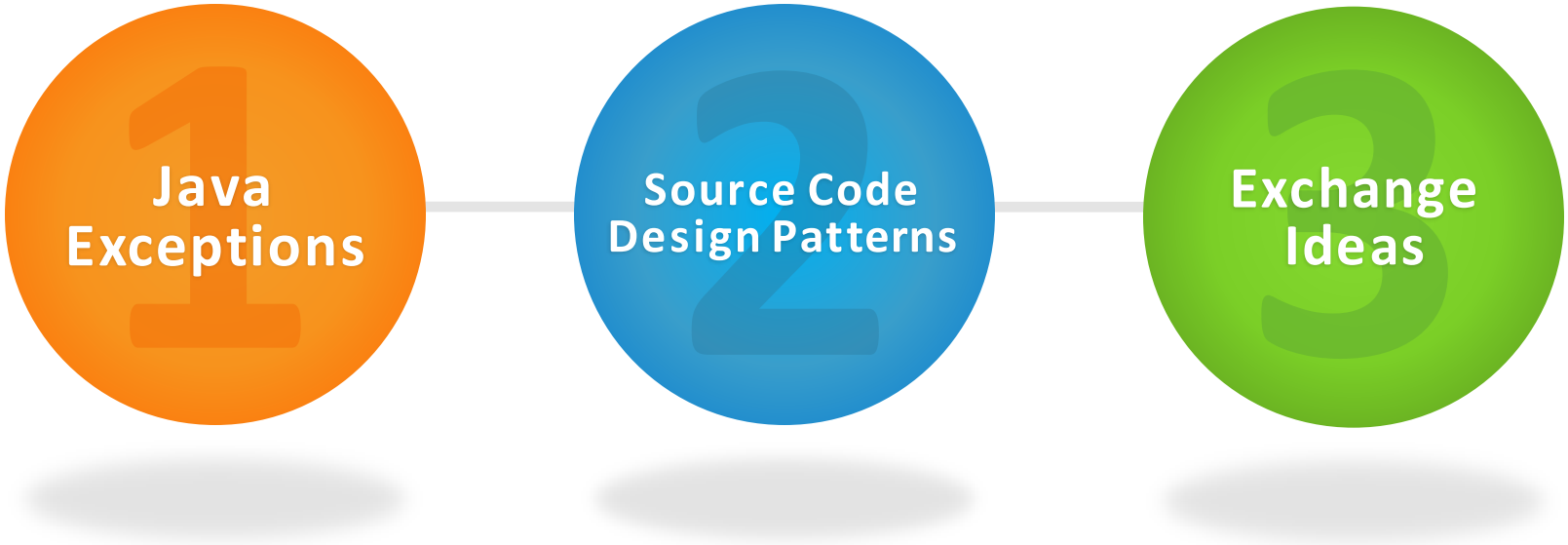
Java Programming – Software App Development

Cristian Toma

D.I.C.E/D.E.I.C – Department of Economic Informatics & Cybernetics
www.dice.ase.ro

Cristian Toma – Business Card

Agenda for Lecture 6 – Summary of JSE

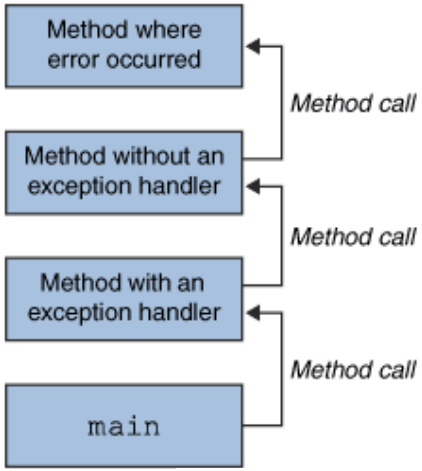


Exception mechanisms and features

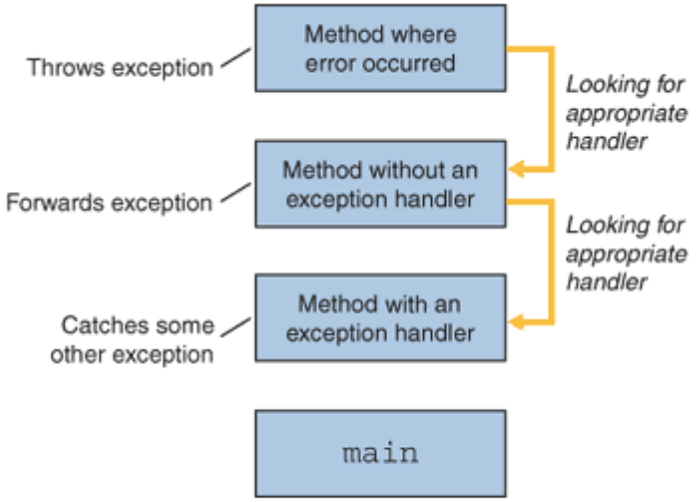
Java Exceptions

1.1 Java Exceptions Summary

Propagation Mode:



The call stack.



Searching the call stack for the exception handler.

1.1 Java Exceptions Summary

Exceptions Types:

1. checked exception

They are not passing by the compilation phase. May exist a “recovery” mechanism but it is a MUST to have “try-catch” source code statements.

2. errors

They are passing by the compilation phase, BUT it is impossible to forecast malfunctions of HW or OS – e.g. HDD has bad sectors and for opening a file there is a ‘java.io.IOError’ throw. In practice, there is not a try-catch statement for them.

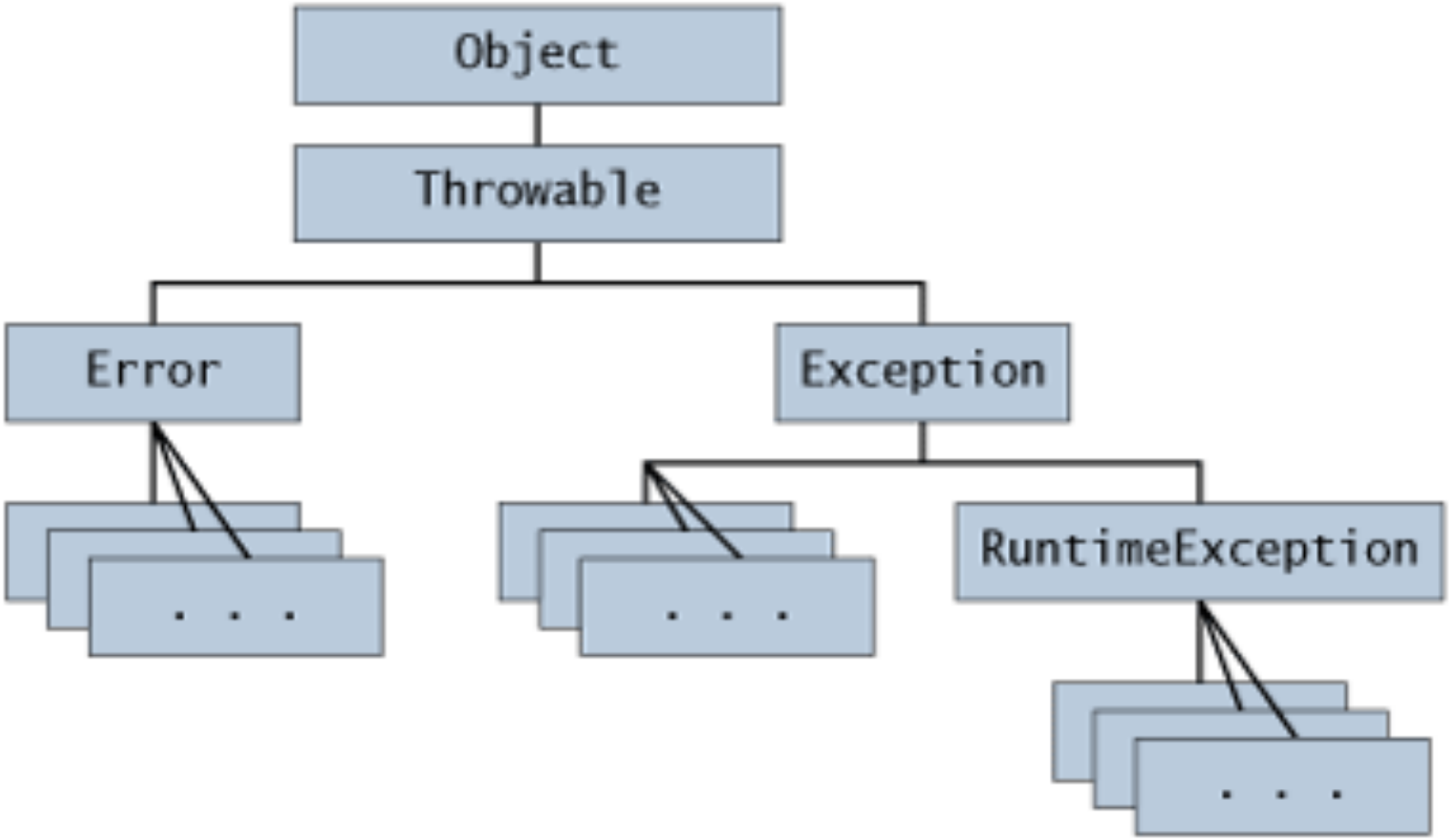
3. runtime exception

They are passing by the compilation phase, BUT the development logics is not implemented correct – e.g. after computations there is a «divison by zero». It is possible to use try-catch mechanism but it is better to investigate and to correct the «logic bug».

- **2+3 = unchecked exception**

1.1 Java Exceptions Summary

Exceptions Class Hierarchy in Java:



1.1 Java Exceptions Summary

Exceptions C vs. Java/C++ approach:

```
errorCodeType readFile {
    initialize errorCode = 0;

    open the file;
    if (theFileIsOpen) {
        determine the length of the file;
        if (gotTheFileLength) {
            allocate that much memory;
            if (gotEnoughMemory) {
                read the file into memory;
                if (readFailed) {
                    errorCode = -1;
                }
            } else {
                errorCode = -2;
            }
        } else {
            errorCode = -3;
        }
        close the file;
        if (theFileDidntClose && errorCode == 0) {
            errorCode = -4;
        } else {
            errorCode = errorCode and -4;
        }
    } else {
        errorCode = -5;
    }
    return errorCode;
}
```

```
readFile {
    try {
        open the file;
        determine its size;
        allocate that much memory;
        read the file into memory;
        close the file;
    } catch (fileOpenFailed) {
        doSomething;
    } catch (sizeDeterminationFailed) {
        doSomething;
    } catch (memoryAllocationFailed) {
        doSomething;
    } catch (readFailed) {
        doSomething;
    } catch (fileCloseFailed) {
        doSomething;
    }
}
```

Section Conclusion

Fact: **Java Exceptions**

In few **samples** it is simple to remember: Exceptions mechanisms and types in Java.

Source Code Design Patterns – factory methods, singletons

Source Code Design Patterns

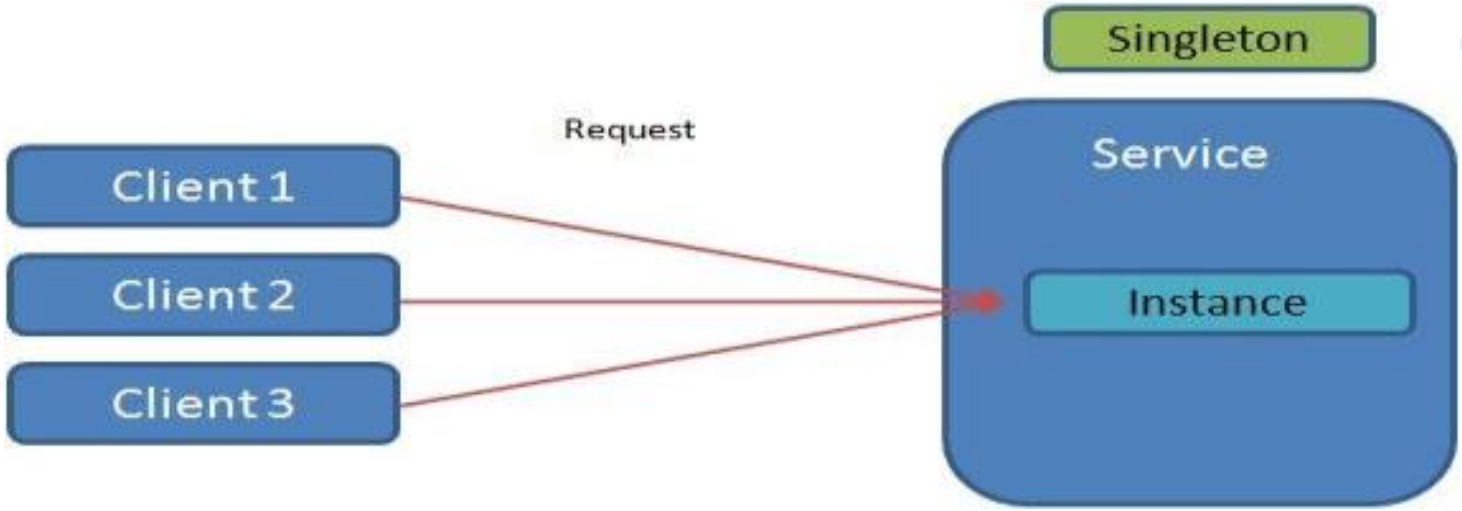
1.2 Java Source Code Design Patterns Summary

Java Singleton:

```
public class SimpleSingleton {
    private static SimpleSingleton singletonInstance = null;
    //Mark the constructor private to avoid object creation outside.
    private SimpleSingleton() {

    }
    //This is where other object can obtain instance of this class.
    public static SimpleSingleton getInstance() {
        if (null == singletonInstance) {
            singletonInstance = new SimpleSingleton();
        }

        return singletonInstance;
    }
```



http://searchdaily.net/category/java/designpattern/creational-pattern/

1.2 Java Source Code Design Patterns Summary

Bruce Eckel, "Thinking in Patterns with Java",
http://www.tutok.sk/fastgl/download/books/Thinking%20in%20Patterns%20with%20Java.pdf
One of the best book for source code design patterns.

Java Singleton:

- ***"Singleton is used to control the amount of created objects."***
- **In same category beside Singleton, there is Objects Pool.**

Java Factory Method:

Where to use & benefits

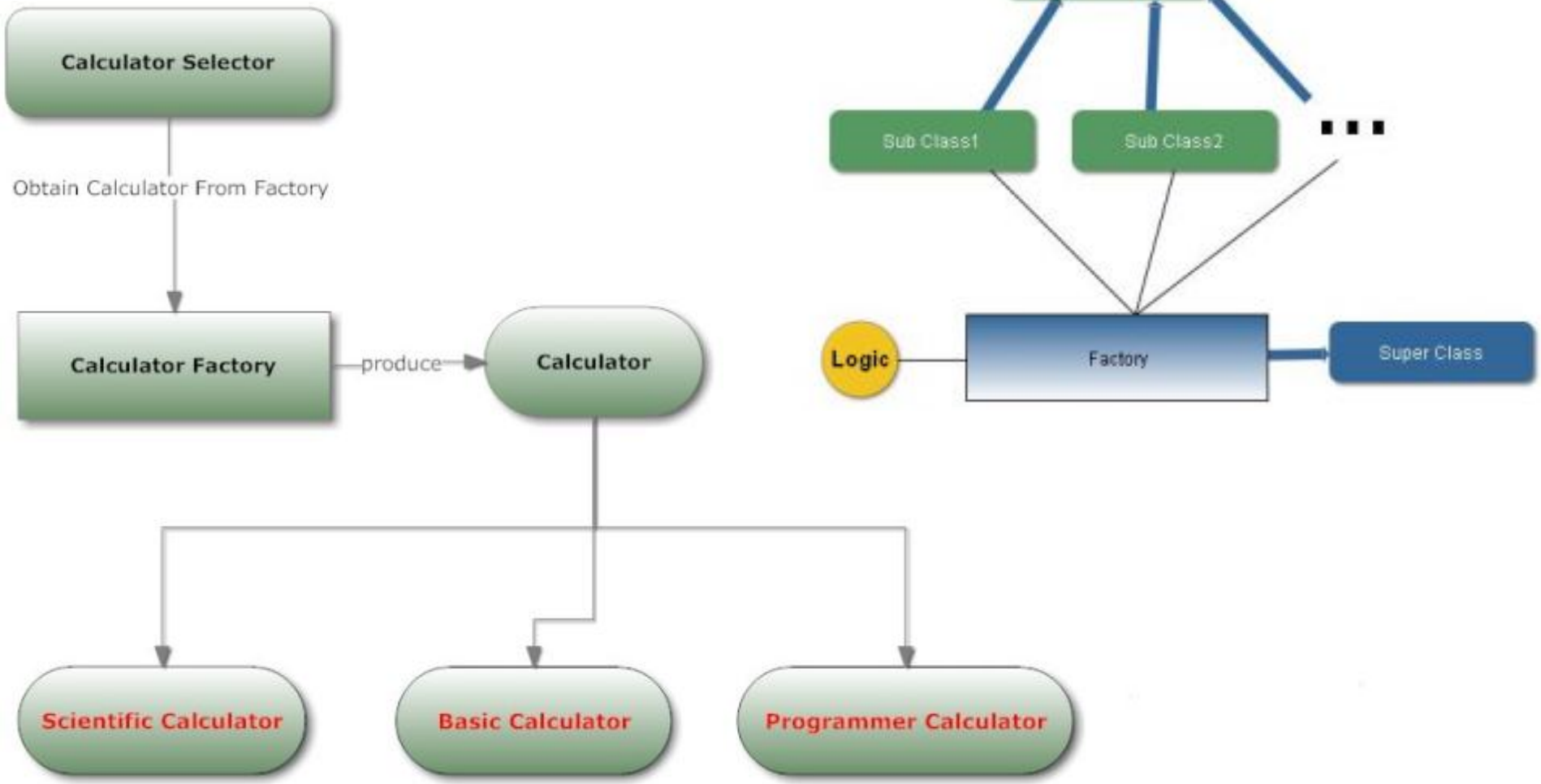
- Connect parallel class hierarchies.
- A class wants its subclasses to specify the object.
- A class cannot anticipate its subclasses, which must be created.
- A family of objects needs to be separated by using shared interface.
- The code needs to deal with interface, not implemented classes.
- Hide concrete classes from the client.
- Factory methods can be parameterized.
- The returned object may be either abstract or concrete object.
- Providing hooks for subclasses is more flexible than creating objects directly.
- Follow naming conventions to help other developers to recognize the code structure.

http://javamagic.wordpress.com/2010/08/27/factory-method-pattern/

1.2 Java Source Code Design Patterns Summary

http://javamagic.wordpress.com/2010/08/27/factory-method-pattern/

Java Factory Method:



http://searchdaily.net/factory-method-pattern-tutorial/

1.2 Java Source Code Design Patterns Summary

Java Factory Method:

```
public class Calculator {
    public Calculator() {

    }

    protected String name;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    protected String type;

    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }
}
```

```
public class ScientificCalculator extends Calculator {
    public ScientificCalculator(String name) {
        System.out.println("Hello I'm " + name);
    }
}
```

```
public class BasicCalculator extends Calculator {
    public BasicCalculator(String name) {
        System.out.println("Hello I'm " + name);
    }
}
```

```
public class ProgrammerCalculator extends Calculator {
    public ProgrammerCalculator(String name) {
        System.out.println("Hello I'm " + name);
    }
}
```

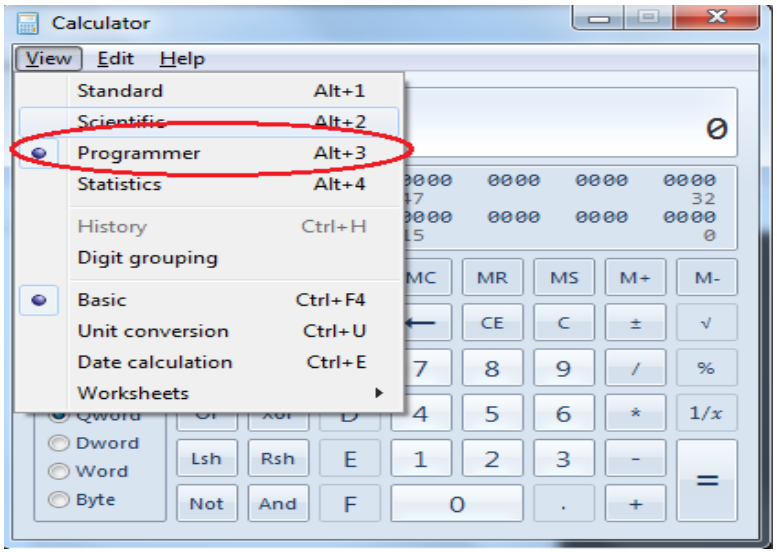
http://searchdaily.net/factory-method-pattern-tutorial/

1.2 Java Source Code Design Patterns Summary

Java Factory Method:

http://searchdaily.net/factory-method-pattern-tutorial/

```
public class CalculatorFactory {
    public Calculator getCalculator(final String type, final String name) {
        if ("B".equals(type)) {
            return new BasicCalculator(name);
        } else if ("S".equals(type)) {
            return new ScientificCalculator(name);
        } else if ("P".equals(type)) {
            return new ProgrammerCalculator(name);
        } else {
            return new Calculator();
        }
    }
}
```




```
public class CalculatorSelector {
    public static void main(String[] args) {
        CalculatorFactory factory = new CalculatorFactory();
        Calculator calculator1 = factory.getCalculator("P", "a Programmer Calculator");
        Calculator c2 = factory.getCalculator("B", "a Basic Calculator");
        System.out.println("c1 type: " + calculator1.getClass().getName());
        System.out.println("c2 type: " + c2.getClass().getName());
    }
}
```

```
<terminated> CalculatorSelector [Java Application] C:\Java\jre6\bin\javaw.exe (Aug 6, 2011 11:32:52 AM)
Hello I'm a Programmer Calculator
Hello I'm a Basic Calculator
c1 type: net.searchdaily.java.design.pattern.factorymethod.ProgrammerCalculator
c2 type: net.searchdaily.java.design.pattern.factorymethod.BasicCalculator
```

Section Conclusions

Source code design patterns such as: Singleton, Objects Pool, Factory Methods…patterns used in any kind of software solution.

Source code design patterns
for easy sharing

Share knowledge, Empowering Minds

Communicate & Exchange Ideas

Questions & Answers!

Recapitulation with samples…

From previous lectures!

But wait…

There’s More!

Thanks!

Java SE Programming
End of Lecture 5 – summary of Java SE