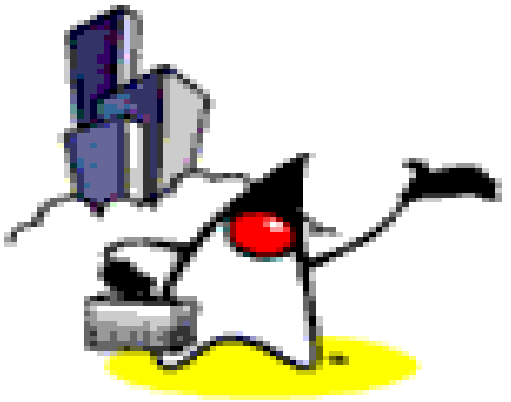
# Java AOP Basics

Sang Shin
Michèle Garoche
www.javapassion.com
"Learning is fun!"

# Topics

- What is and Why AOP?
- AOP terminology
- AspectJ
- Spring AOP



# What is & Why AOP?

# OOP Design is good but..

- The OOP makes the code design more reusable and compact.
- But sometimes it makes the code scattered in number of modules.
- Example
  - adding message logging in each of class methods makes the logging code scattered in number of methods and classes.
  - This makes code maintenance and update a tedious work.

# Issues that do not get solved by OOP

- Code scattering
  - Same concern spread across modules
- Code tangling
  - Coupling of different concerns

# AOP is the solution

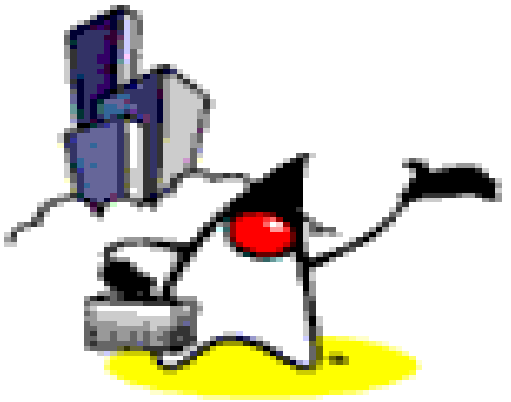
- Aspect Oriented Programming(AOP) addresses these issues.
- Instead of adding the logging code in each of the methods, you delegate the logging functionality (concern) across number of methods to an AOP container and specify which methods require logging applied.
- The container will call the logging aspect of the code when the methods are invoked on the proxy object returned by the Container.

# AOP and OOP

- Aspect-Oriented Programming (AOP) complements Object-Oriented Programming (OOP) by providing another way of thinking about program structure.
- Aspects enable modularization of concerns (required additional business logic) such as transaction management that cut across multiple types and objects

# AOP Benefits

- Complements OO programming.
- Clean and modular code without additional code plumbing.
- Aspects can be added or removed as needed without changing the code.
- The business object code is not tied to specific API or framework



# AOP Terminologies

# Cross-cutting Concerns

- The required functionality
- Examples of cross-cutting concerns
  - Logging
  - Transaction management
  - Security
  - Auditing
  - Locking
  - Event handling

# AOP Concepts: Aspects

- A modularization of a concern that cuts across multiple objects and methods.

# AOP Concepts: Advice

- The code that is executed at a particular joinpoint
- Types of Advice
  - before advice, which executes before joinpoint
  - after advice, which executes after joinpoint
  - around advice, which executes around joinpoint

# AOP Concepts: Join point

- A Join point is a point in the execution of a program
- Example
  - Method invocation
  - Exception handing
- In Spring AOP, a join point always represents a method execution.

# AOP Concepts: Pointcuts

- A pointcut is an expression that selects one or more join points
- By creating pointcuts, you gain fine-grained control over how you apply advice to the components
- Example
  - A typical joinpoint is a method invocation.
  - A typical pointcut is a collection of all method invocations in a particular class
- Pointcuts can be composed in complex relationships to further constrain when advice is executed

# AOP Concepts: Target

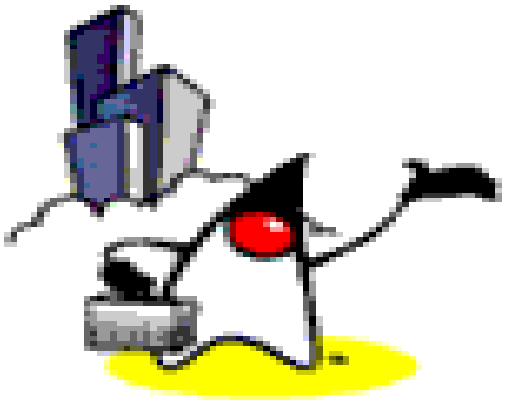
- An object whose execution flow is modified by some AOP process
- They are sometimes called advised object

# AOP Concepts: Weaving

- Process of actually inserting aspects into the application code at the appropriate point
- Types of Weaving
  - Compile time weaving
  - Runtime weaving

# AOP Concepts: Introduction

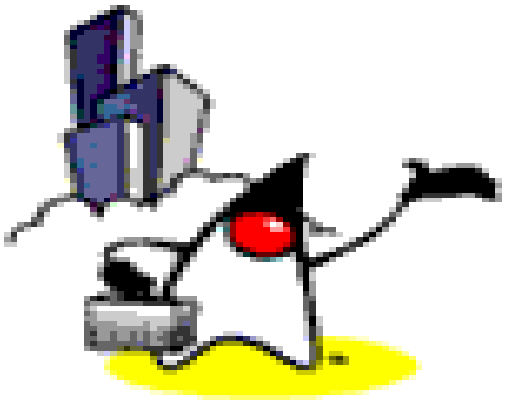
- Process by which you can modify the structure of an object by introducing additional methods or fields to it
- You use the *Introduction* to make any object implement a specific interface without needing the object's class to implement that interface explicitly



# Types of AOP

# Types of AOP

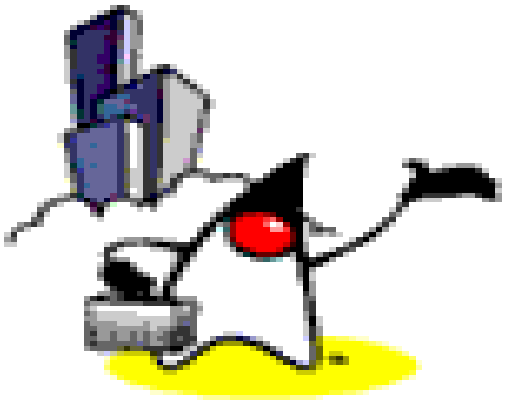
- Static AOP
  - The weaving process forms another step in the build process for an application
  - Example: In Java program, you can achieve the weaving process by modifying the actual bytecode of the application changing and modifying code as necessary
- Dynamic AOP
  - The weaving process is performed dynamically at runtime
  - Easy to change the weaving process without recompilation



# AOP Tools

# AOP Tools

- AspectJ - AOP extension to Java
- Spring AOP - AOP extension to Spring framework
- ...



# AspectJ

# Steps to follow

- Write target class
- Write aspect
  - Define joint point
  - Write advice
- Write wrapper/testing code
- Compile aspect and Java code using “ajc”
- Run the application

# 1. Write Target class

- HelloWorld.java

```
// Target class - HelloWorld.java
public class HelloWorld {
    public static void say(String message) {
        System.out.println(message);
    }

    public static void sayToPerson(String message, String name) {
        System.out.println(name + ", " + message);
    }
}
```

# 2. Write Aspect

- MannersAspect.java

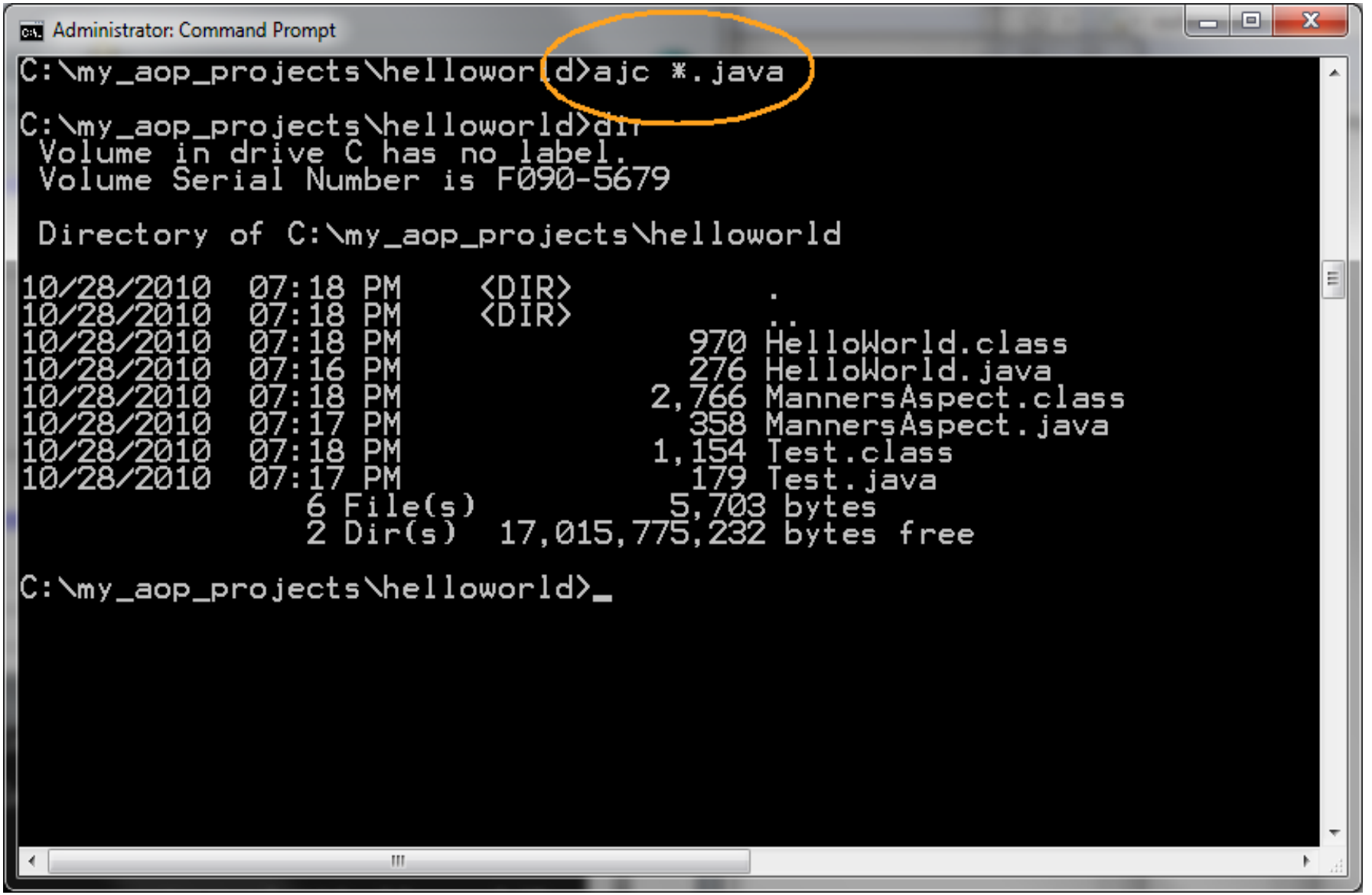
```
// Aspect - MannersAspect.java
public aspect MannersAspect {

    // pointcut declares that what follows is a declaration of a named pointcut.
    // Next, callSayMessage(), the pointcut's name, resembles a method
declaration.
    // The trailing empty () suggests that the pointcut collects no context.

    pointcut callSayMessage() : call(public static void HelloWorld.say*(..));

    // Advice #1
    before() : callSayMessage() {
        System.out.println("Good day!");
    }

    // Advice #2
    after() : callSayMessage() {
        System.out.println("Thank you!");
    }
}
```

# 3. Write Wrapper/Testing Code

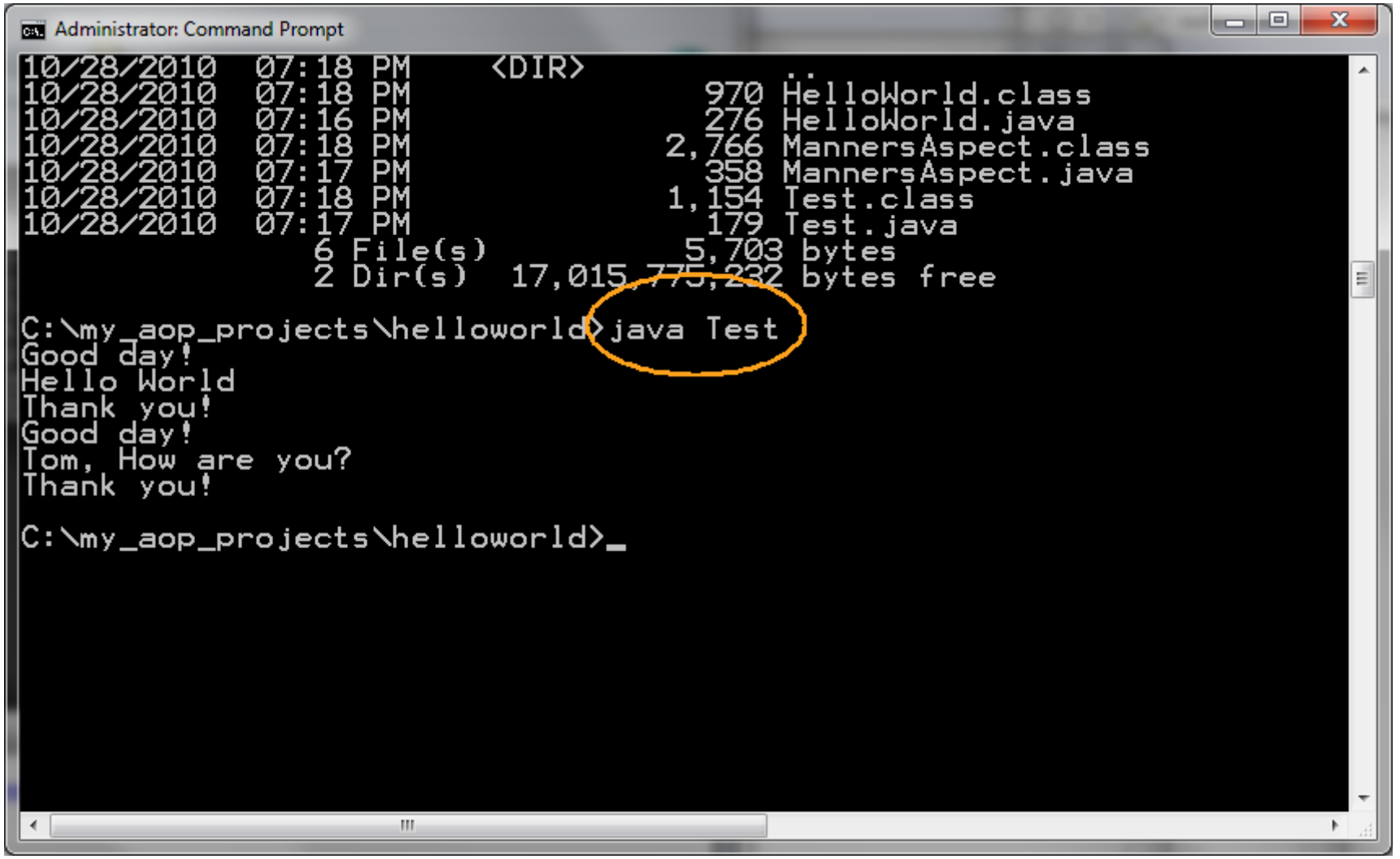
- Test.java

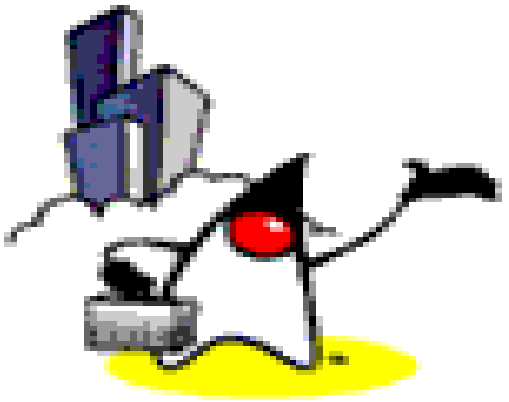
```
// Test code
public class Test {
    public static void main(String[] args) {
        HelloWorld.say("Hello World");
        HelloWorld.sayToPerson("How are you?", "Tom");
    }
}
```

# 4. Compile with “ajc”



# 5. Run the application

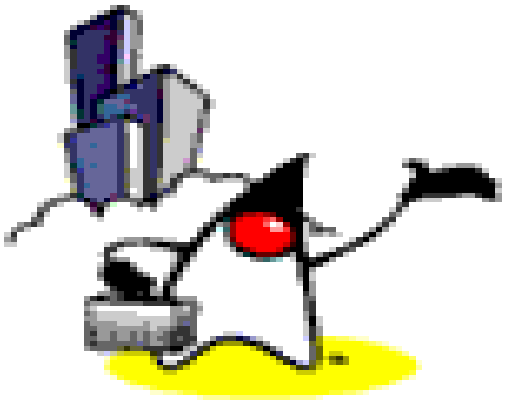




# Spring AOP

# Spring AOP

- AOP Framework that builds on the aopalliance interfaces.
- Aspects are coded with pure Java code.
- Based on proxies
  - When you want to create an advised instance of a class, you must use the *ProxyFactory* class to create a proxy of an instance of that class, first providing the *ProxyFactory* with all the aspects that you want to be woven into the proxy
  - You typically use *ProxyFactoryBean* class to provide declarative proxy creation



# HelloWorld Spring AOP

# MessageWriter Class -Target Class

- The target class just displays “World”
- The joinpoint is the invocation of the *writeMessage*() method
- We want to display “Life is Good indeed!” through AOP – we need “around” advice

```
// This is a target class
public class MessageWriter {
    public void writeMessage() {
        System.out.print("Good");
    }
}
```

# MessageDecorator - Around Advice

- *MethodInterceptor* is AOP Alliance standard interface for around interface
- *MethodInvocation* object represents the method invocation that is being advised

```
public class MessageDecorator implements
  MethodInterceptor {

    public Object invoke(MethodInvocation invocation)
                                    throws Throwable {
        System.out.print("Life is ");
        Object retVal = invocation.proceed();
        System.out.println(" indeed!");
        return retVal;
    }
}
```

# Weaving MessageDecorator Advice

- Use *ProxyFactory* class to create the proxy of the target object

```
public static void main(String[] args) {
        // Create a target
        MessageWriter target = new MessageWriter();

        // Create the proxy factory
        ProxyFactory pf = new ProxyFactory();

        // Add a given advice to the tail
        // of the advice (interceptor) chain
        pf.addAdvice(new MessageDecorator());
```

# Weaving MessageDecorator Advice

```
        // Set the given object as target
        pf.setTarget(target);

        // Create a proxy according to the
        // settings in the proxy factory
        MessageWriter proxy =
                (MessageWriter) pf.getProxy();

        // use the proxy
        proxy.writeMessage();
    }
}
```

# Thank you!

Sang Shin
Michèle Garoche
http://www.javapassion.com
“Learning is fun!”