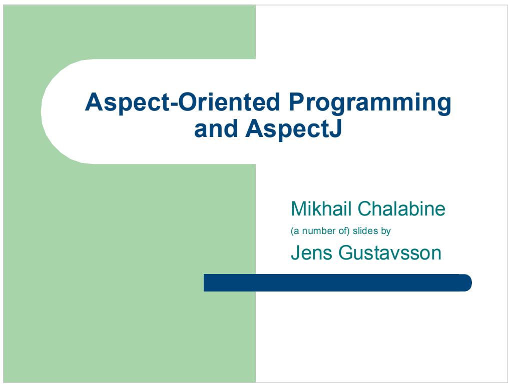
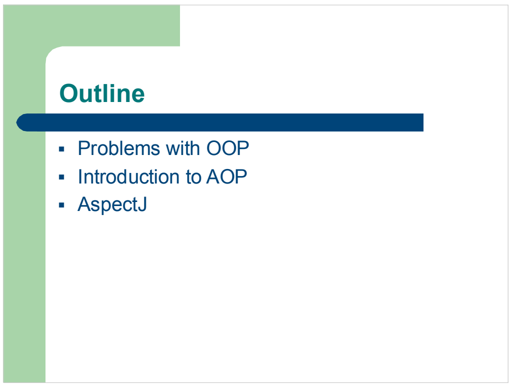


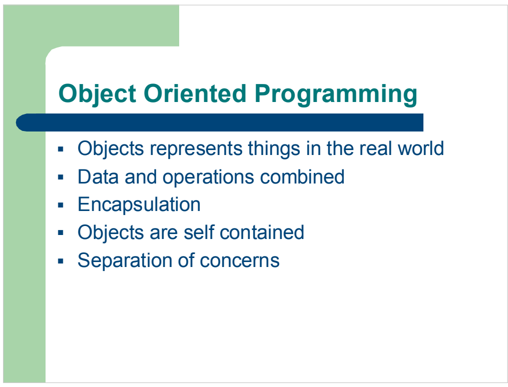
Aspect-Oriented Programming and AspectJ

Mikhail Chalabine
(a number of) slides by
Jens Gustavsson



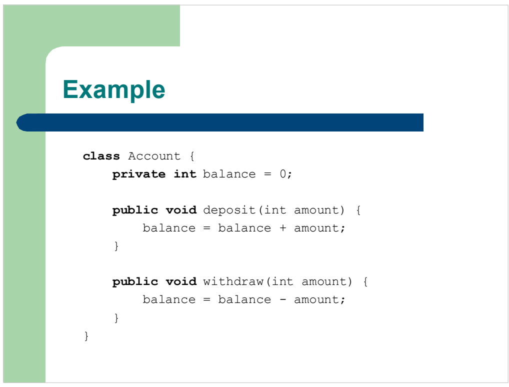
Outline

- Problems with OOP
- Introduction to AOP
- AspectJ



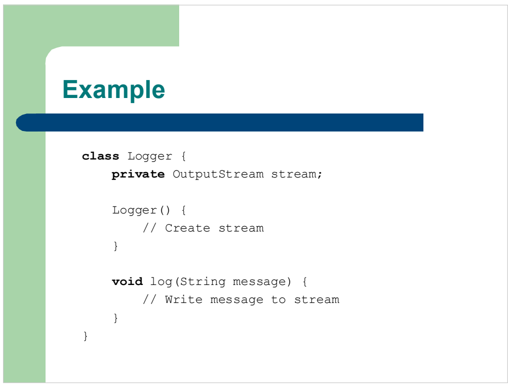
Object Oriented Programming

- Objects represents things in the real world
- Data and operations combined
- Encapsulation
- Objects are self contained
- Separation of concerns



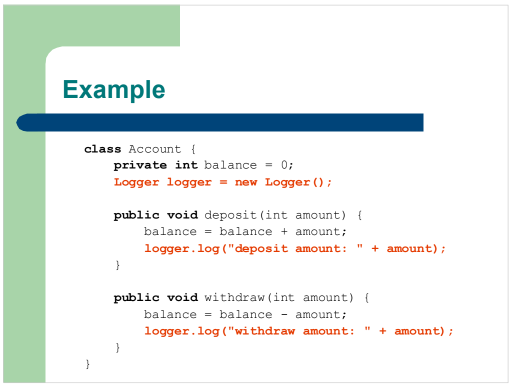
Example

```
class Account {
    private int balance = 0;

    public void deposit(int amount) {
        balance = balance + amount;
    }

    public void withdraw(int amount) {
        balance = balance - amount;
    }
}
```



Example

```
class Logger {
    private OutputStream stream;

    Logger() {
        // Create stream
    }

    void log(String message) {
        // Write message to stream
    }
}
```



Example

```
class Account {
    private int balance = 0;
    Logger logger = new Logger();

    public void deposit(int amount) {
        balance = balance + amount;
        logger.log("deposit amount: " + amount);
    }

    public void withdraw(int amount) {
        balance = balance - amount;
        logger.log("withdraw amount: " + amount);
    }
}
```

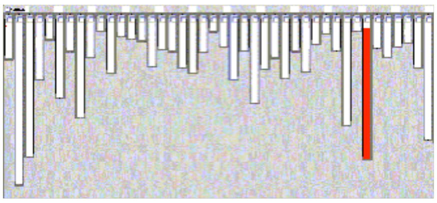
Crosscutting

- Code in objects that does not relate to the functionality defined for those objects.
- Imagine adding:
  - User authentication
  - Persistence
  - Timing
  - …
- Mixing of concerns lead to:
  - Code scattering
  - Code tangling

Mixing Concerns

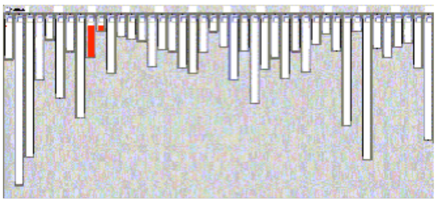
- Correctness
  - Understandability
  - Testability
- Maintenance
  - Find code
  - Change it consistently
  - No help from OO tools
- Reuse

XML parsing



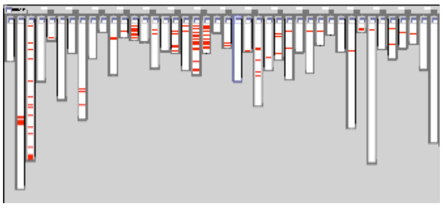
- XML parsing in org.apache.tomcat
  - red shows relevant lines of code
  - nicely fits in one box

URL pattern matching



- URL pattern matching in org.apache.tomcat
  - red shows relevant lines of code
  - nicely fits in two boxes (using inheritance)

logging is not modularized

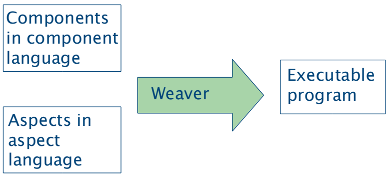


- logging in org.apache.tomcat
  - red shows lines of code that handle logging
  - not in just one place
  - not even in a small number of places

Aspect Oriented Programming

- Aspect = Concern that crosscuts other components.
  A more precise definition comes later!
- Components written in *component language*
- Provide a way to describe aspects in *aspect language*
- Not to replace OOP
- Does not have to be OO based

Aspect Weaving



Weaving Time

- Preprocessor
- Compile time
- Link time
- Load time
- Run time

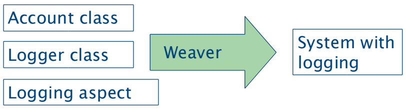
Example

```
class Account {
    private int balance = 0;

    public void deposit(int amount) {
        balance = balance + amount;
    }

    public void withdraw(int amount) {
        balance = balance - amount;
    }
}
```

Example (*ad hoc* syntax)

```
define aspect Logging {

    Logger logger = new Logger();

    when calling any method(parameter "amount") {
        logger.log(methodname + " amount: " + amount);
    }
}
```

Aspect Weaving



Concepts added by AOP Languages

- Join points
- Pointcuts
- Advice
- Aspects
- Weaving

Join Point

- A location in (component) code where a concern crosscuts (static join point model)
- A well-defined point in the program flow (dynamic join point model, e.g., in AspectJ)
- Examples:
  - Method / class declaration
  - A call to a method
  - etc.

```
public void Account.deposit(int)
```

Pointcut

- A pointcut picks out certain join points and values at those points
  - Specifies when a join point should be matched
- In the followin the `balanceAltered` pointcut picks out each join point that is a call to either the `deposit()` or the `withdraw()` method of an `Account` class

```
pointcut balanceAltered() :
  call(public void Account.deposit(int)) ||
  call(public void Account.withdraw(int));
```

Pointcut (further examples)

- `call(void SomeClass.make*(..))`
  - picks out each join point that's a call to a void method defined on `SomeClass` whose the name begins with "make" regardless of the method's parameters
- `call(public * SomeClass.* (..))`
  - picks out each call to `SomeClass`e's public methods
- `cflow(somePointcut)`
  - picks out each pointcut that occurs in the dynamic context of the join points picked out by `somePointcut`
  - pointcuts in the control flow, e.g., in a chain of method calls

A piece of Advice

- Code that is executed at a pointcut (when a join point is reached)

```
before(int i) : balanceAltered(i) {
  System.out.println("The balance changed");
}
```

Aspect

- Groups join points, pointcuts and advice.
- **The unit of modularity for a crosscutting concern.**

```
public aspect LoggingAspect {
  pointcut balanceAltered() :
    call(public void Account.deposit(int)) ||
    call(public void Account.withdraw(int));

  before(int i) : balanceAltered(i) {
    System.out.println("The balance changed");
  }
}
```

Take a breath ... so far we have

- Agreed that *tangled*, *scattered* code that appears as a result of *mixing* different *crosscutting concerns* in (OO) programs is a problem
- Sketched a feasible solution - AOP
- Introduced
  - Join points
  - Pointcuts
  - Advice
  - Aspects
  - Weaving
- Tools?

AspectJ

- Xerox Palo Alto Research Center
- Gregor Kiczales, 1997
- Goal: Make AOP available to many developers
  - Open Source
  - Tool integration Eclipse
- Components in Java
- Java with extensions for describing aspects
- Current focus: industry acceptance

AspectJ Demo

Join Points

- Method call execution
- Constructor call execution
- Field get
- Field set
- Exception handler execution
- Class/object initialization

Patterns

- Match any type: *
- Match 0 or more characters: *
- Match 0 or more parameters: (..)
- `call(private void Person.set*(*)`
- `call(* * *.*(*)`
- `call(* * *.*(..)`
- All subclasses: `Person+`

Logical Operators

- `call((Person+ && ! Person).new(..))`

Example

```
pointcut balanceAccess() :
  get(private int Account.balance);

before() : balanceAccess() {
  System.out.println("balance is
 accessed");
}
```

Exposing Context in Pointcuts

- Improves decision process
- AspectJ gives code access to some of the context of the join point
- Two ways

Exposing Context in Pointcuts

- `thisJoinPoint` class and its methods
- Designators
  - State-based: `this, target, args`
  - Control Flow-based: `cflow, cflowbelow`
  - Class-initialization: `staticinitialization`
  - Program Text-based: `withincode, within`
  - Dynamic Property-based: `If, adviceexecution`

Exposing Context in Pointcuts `thisJoinPoint` Methods

- `getThis()`
- `getTarget()`
- `getArgs()`
- `getSignature()`
- `getSourceLocation()`
- `getKind()`
- `toString()`
- `toShortString()`
- `toLongString()`

Exposing Context in Pointcuts `thisJoinPoint` Methods Example

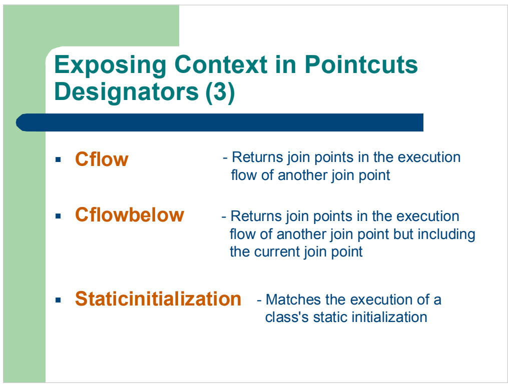
```
public class DVD extends Product {
   private String title;
   ...
}

SourceLocation sl = thisJoinPoint.getSourceLocation();
Class theClass = (Class) sl.getWithinType();
System.out.println(theClass.toString());

Output: class DVD
```

Exposing Context in Pointcuts Designators (1)

- **Execution** - Matches execution of a method or constructor
- **Call** - Matches calls to a method
- **Initialization** - Matches execution of the first constructor
- **Handler** - Matches exceptions
- **Get** - Matches the reference to a class attribute
- **Set** - Matches the assignment to a class attribute

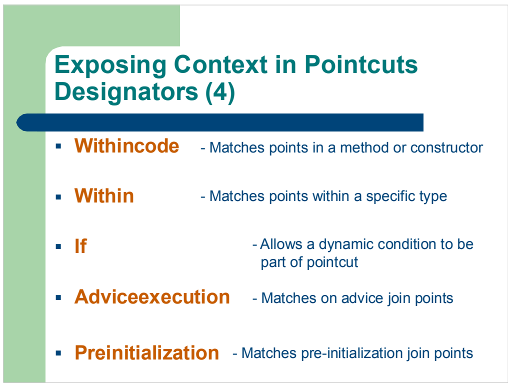
Exposing Context in Pointcuts Designators (2)

- **This** - Returns the object associated with a particular join point or limits the scope of a join point by using a class type
- **Target** - Returns the target object of a join point or limits the scope of join point
- **Args** - Exposes the arguments to a join point or limits the scope of the pointcut



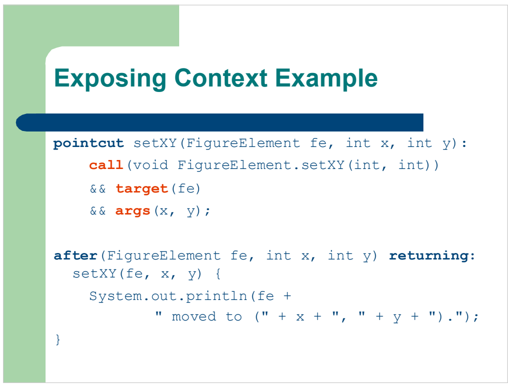
Exposing Context in Pointcuts Designators (3)

- **Cflow** - Returns join points in the execution flow of another join point
- **Cflowbelow** - Returns join points in the execution flow of another join point but including the current join point
- **Staticinitialization** - Matches the execution of a class's static initialization



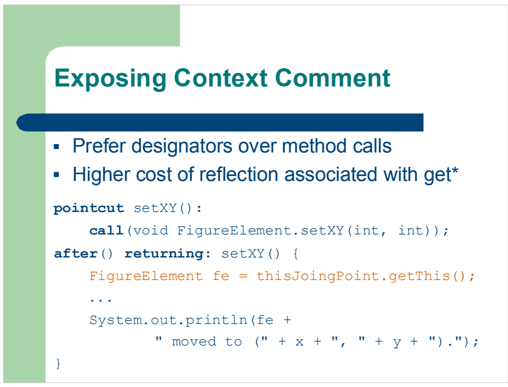
Exposing Context in Pointcuts Designators (4)

- **Withincode** - Matches points in a method or constructor
- **Within** - Matches points within a specific type
- **If** - Allows a dynamic condition to be part of pointcut
- **Adviceexecution** - Matches on advice join points
- **Preinitialization** - Matches pre-initialization join points



Exposing Context Example

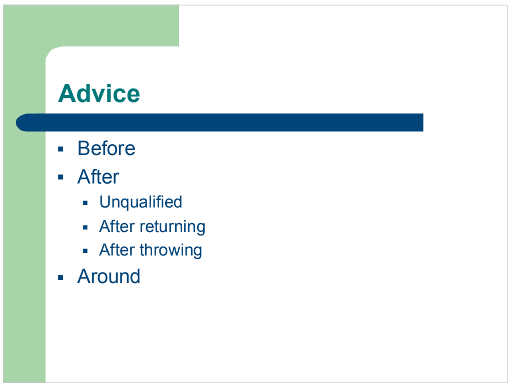
```
pointcut setXY(FigureElement fe, int x, int y):
    call(void FigureElement.setXY(int, int))
    && target(fe)
    && args(x, y);

after(FigureElement fe, int x, int y) returning:
  setXY(fe, x, y) {
    System.out.println(fe +
           " moved to (" + x + ", " + y + ").");
}
```



Exposing Context Comment

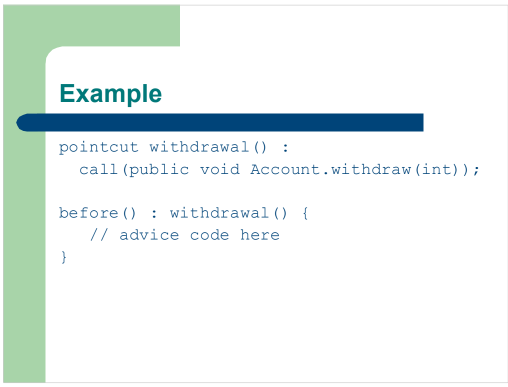
- Prefer designators over method calls
- Higher cost of reflection associated with get*

```
pointcut setXY():
    call(void FigureElement.setXY(int, int));
after() returning: setXY() {
    FigureElement fe = thisJoingPoint.getThis();
    ...
    System.out.println(fe +
           " moved to (" + x + ", " + y + ").");
}
```



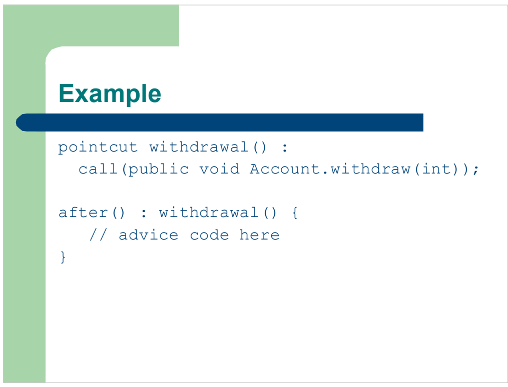
Advice

- Before
- After
  - Unqualified
  - After returning
  - After throwing
- Around



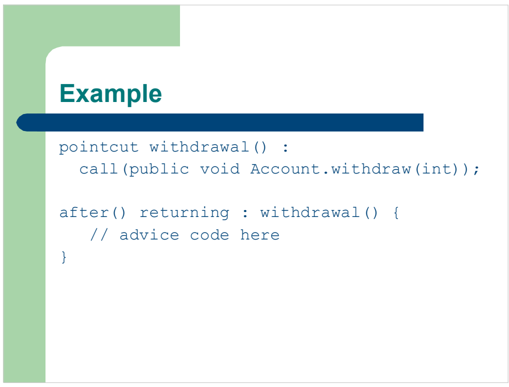
Example

```
pointcut withdrawal() :
  call(public void Account.withdraw(int));

before() : withdrawal() {
   // advice code here
}
```



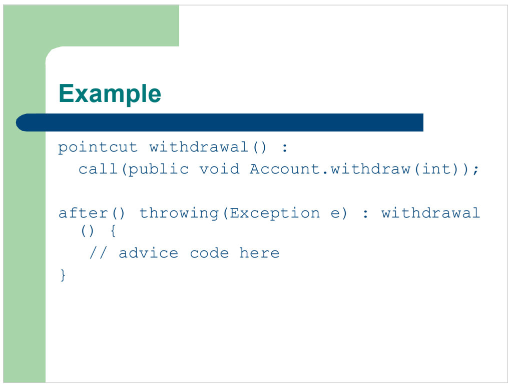
Example

```
pointcut withdrawal() :
  call(public void Account.withdraw(int));

after() : withdrawal() {
   // advice code here
}
```



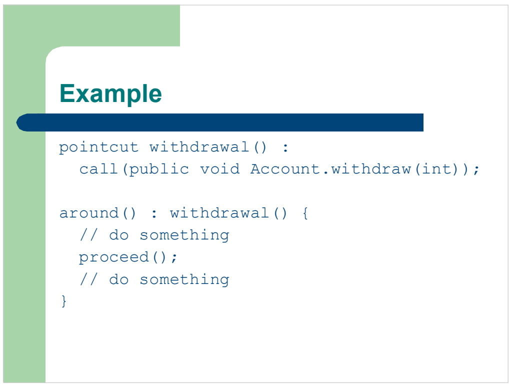
Example

```
pointcut withdrawal() :
  call(public void Account.withdraw(int));

after() returning : withdrawal() {
   // advice code here
}
```



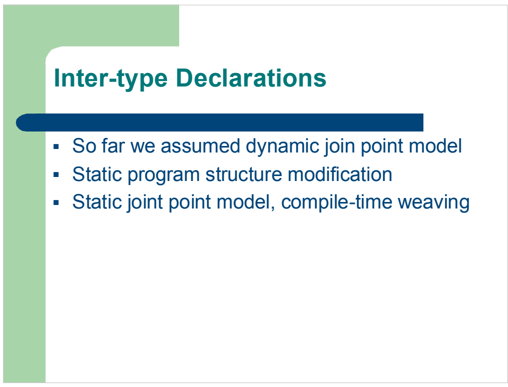
Example

```
pointcut withdrawal() :
  call(public void Account.withdraw(int));

after() throwing(Exception e) : withdrawal
  () {
   // advice code here
}
```



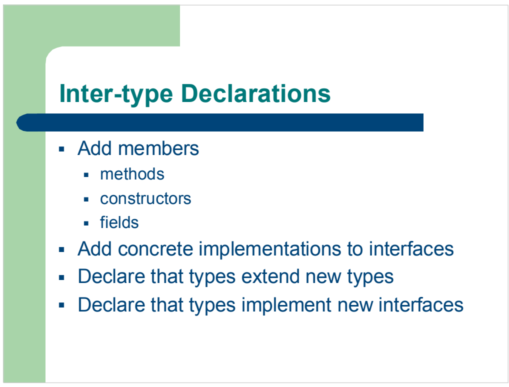
Example

```
pointcut withdrawal() :
  call(public void Account.withdraw(int));

around() : withdrawal() {
  // do something
  proceed();
  // do something
}
```



Inter-type Declarations

- So far we assumed dynamic join point model
- Static program structure modification
- Static joint point model, compile-time weaving



Inter-type Declarations

- Add members
  - methods
  - constructors
  - fields
- Add concrete implementations to interfaces
- Declare that types extend new types
- Declare that types implement new interfaces

Inter-type Declarations Demo

Other AOP languages

- AspectWerkz
- JAC
- JBoss-AOP
- Aspect#
- LOOM.NET
- AspectR
- AspectS
- AspectC
- AspectC++
- Pythius

AOP Brainstorming Examples

- Resource pooling connections
- Caching
- Authentication
- Design by contract
- Wait cursor for slow operations
- Inversion of control
- Runtime evolution

Aspect-Oriented Programming and AspectJ

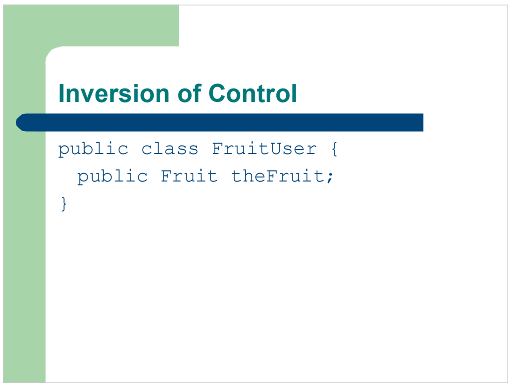
Questions & Answers

Aspect Instantiation

- Aspects are converted to classes by AspectJ compiler
- Types of instantiation:
  - Singleton
  - Per-object
  - Per-control-flow
- Aspects can contain fields (and methods)

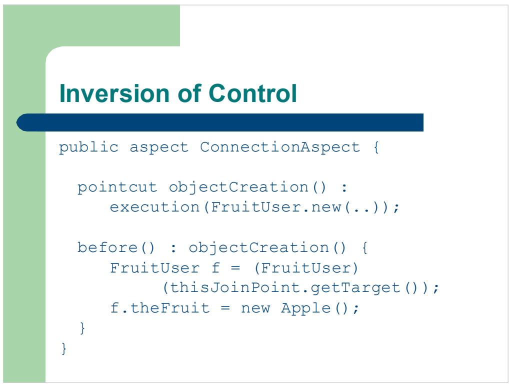
Inversion of Control

```
public class Fruit {}

public class Apple extends Fruit {
  public String toString() {
     return "I am an apple";
  }
}
```



Inversion of Control

```
public class FruitUser {
  public Fruit theFruit;
}
```



Inversion of Control

```
public aspect ConnectionAspect {

  pointcut objectCreation() :
     execution(FruitUser.new(..));

  before() : objectCreation() {
     FruitUser f = (FruitUser)
          (thisJoinPoint.getTarget());
     f.theFruit = new Apple();
  }
}
```