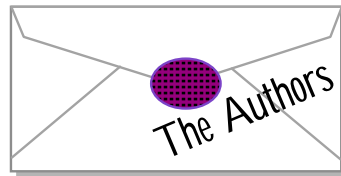


A Case for Class Sealing in Java

Marina Biberstein, IBM Haifa

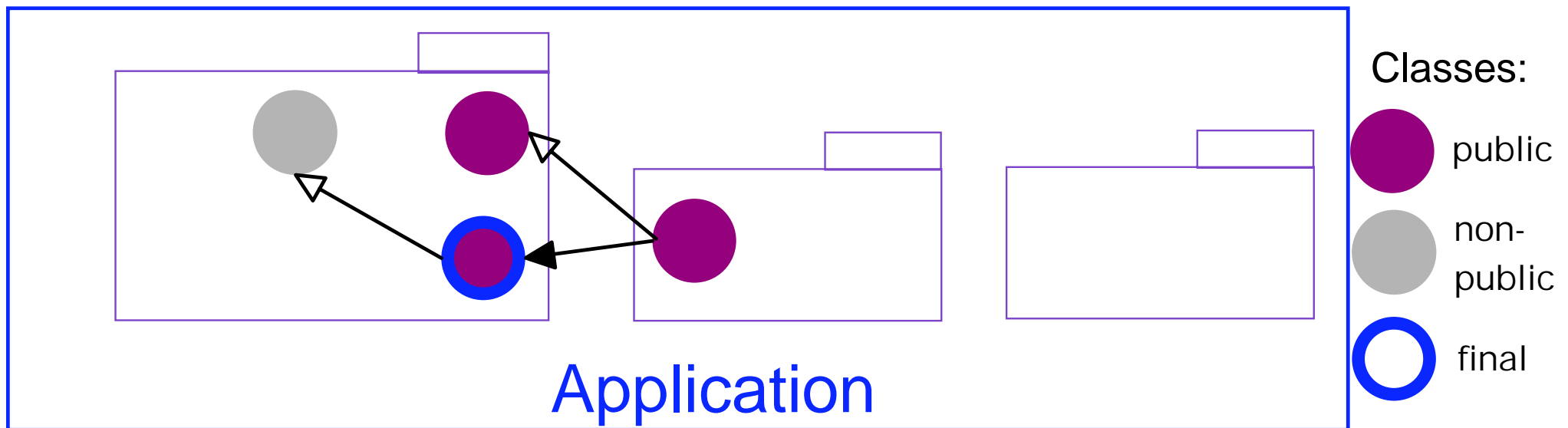
Vugranam C. Sreedhar, IBM Watson

Ayal Zaks, IBM Haifa



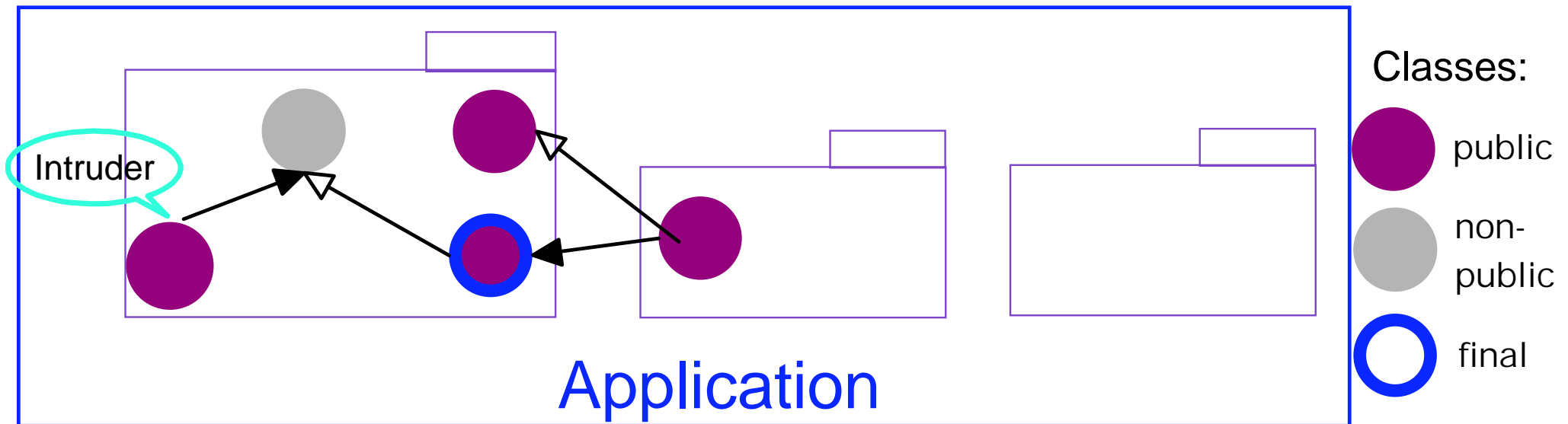
Background: Java Packages

- Software units: application, package, class
- Packages are open
 - Any class can claim package membership
- Packages provide access protection



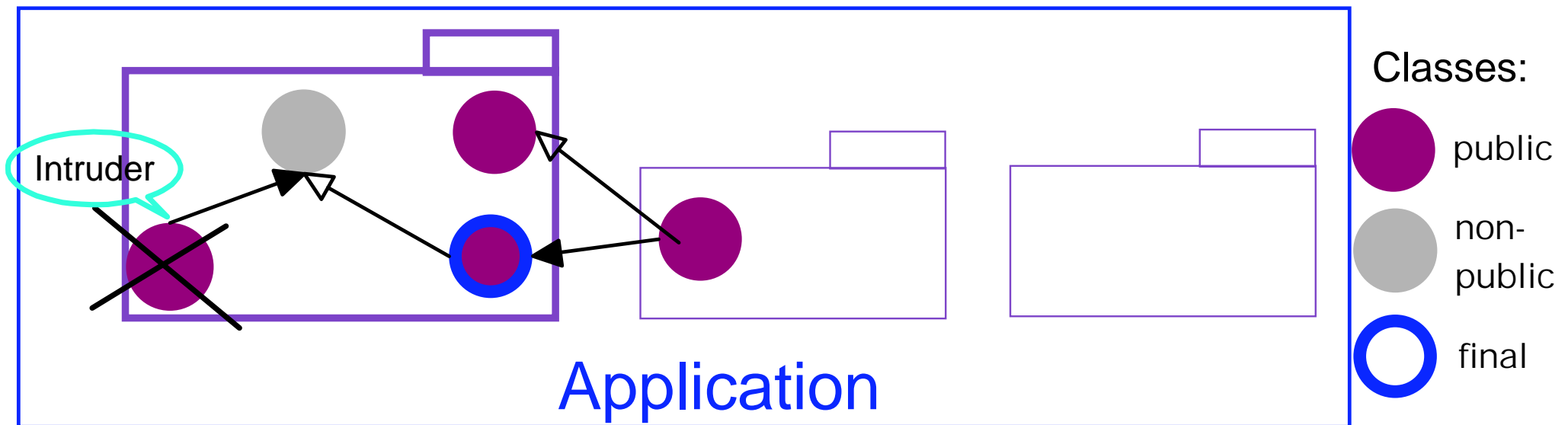
Background: Java Packages

- Software units: application, package, class
- Packages are open
 - Any class can claim package membership
- Packages provide access protection



Package Sealing

- Security risk if a hostile class joins the package
- ✓ Java 2: Sealed packages
 - Java Runtime ensures that all classes in a sealed package originate from the same archive

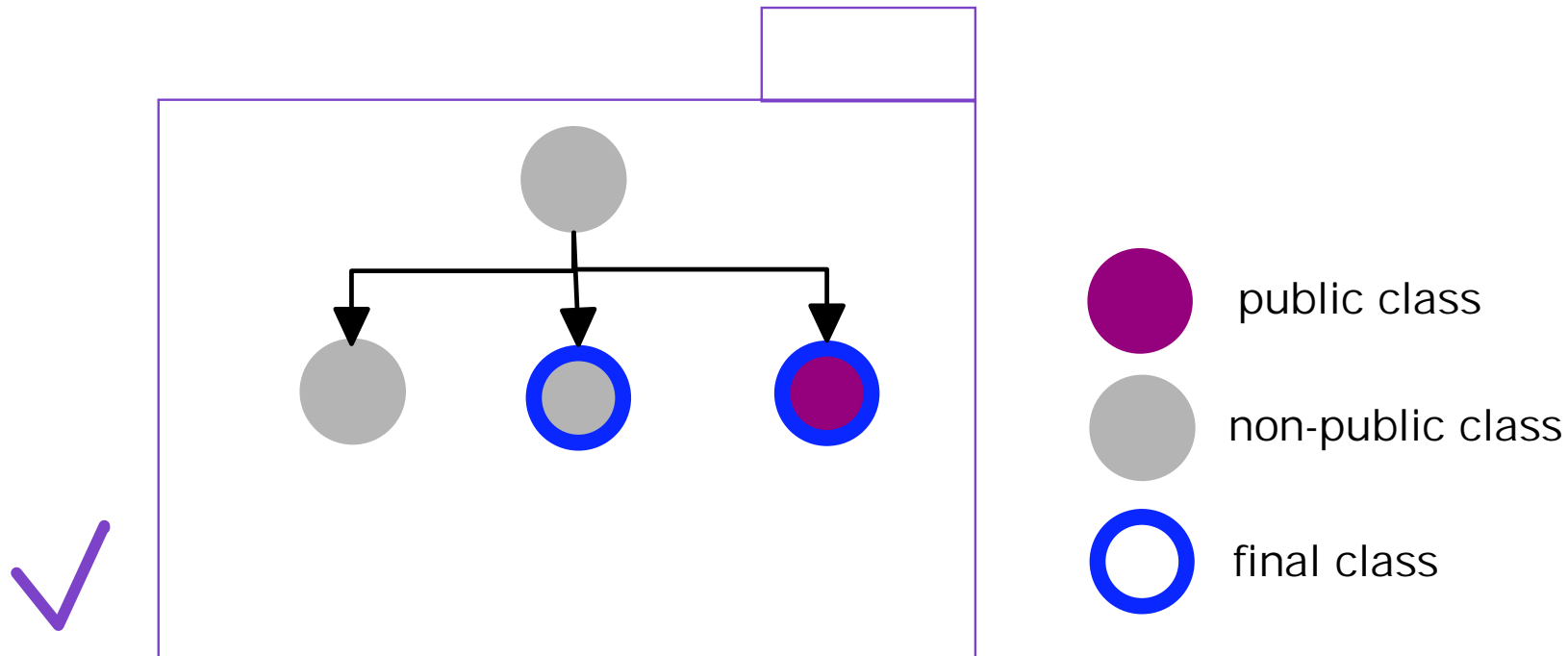


Background: Oligomorphism

- Classes can be:
 - Monomorphic: no subclasses
 - Oligomorphic: set of subclasses is bounded at processing time
 - Polymorphic: unlimited set of subclasses
- Oligomorphism: good for analysis and optimization, security, code understanding, ...

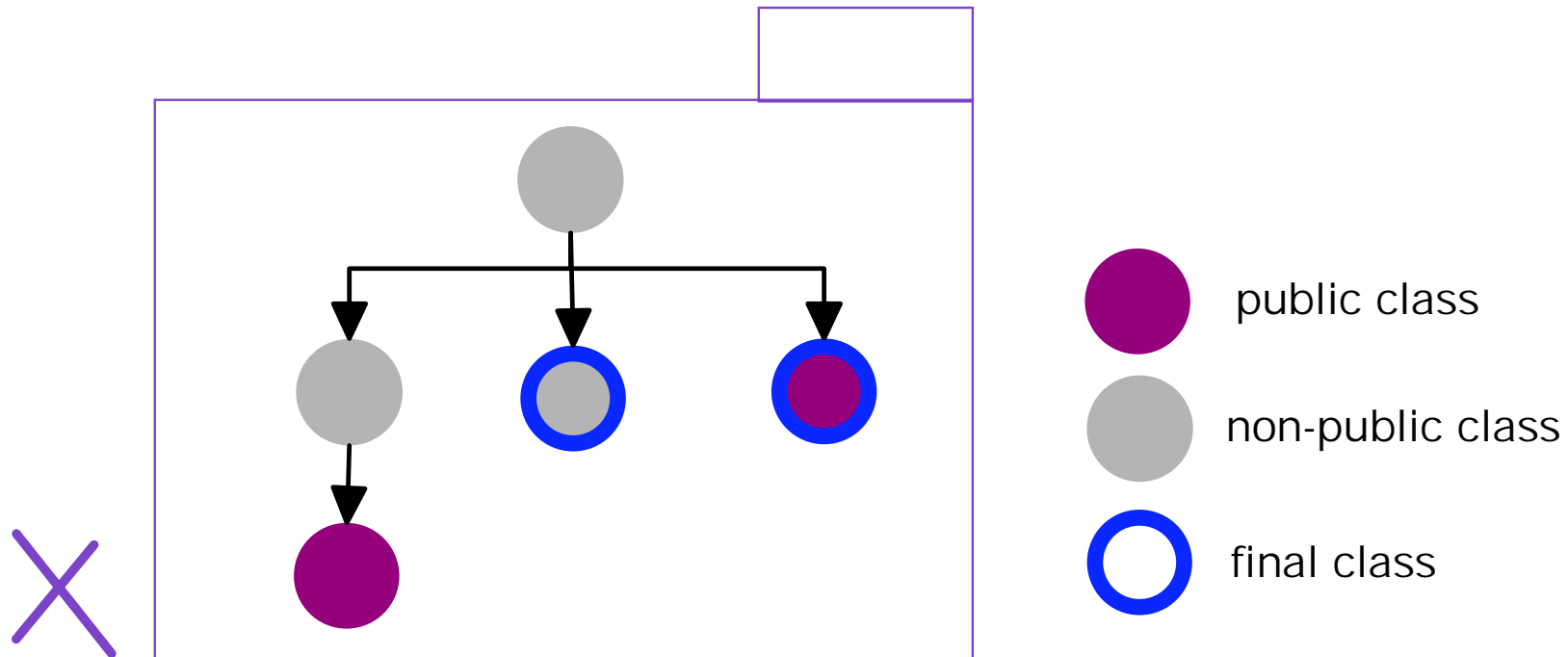
Achieving Oligomorphism in Java

- Make the class final
 - Very restrictive
- Hide class hierarchy within a sealed package
 - Can be broken by mistake



Achieving Oligomorphism in Java

- Make the class final
 - Very restrictive
- Hide class hierarchy within a sealed package
 - Can be broken by mistake



Background: Fragile Base Class

- Modification of a superclass, even if interfaces remain intact, may break correctness of the subclasses

```
class Base{  
    protected int x;  
    protected void m(){  
        x++;  
    }  
    protected void n(){  
        x++;  
    }  
}
```

```
class Sub extends Base{  
    protected void m(){  
        n();  
    }  
}
```


Background: Fragile Base Class

- Modification of a superclass, even if interfaces remain intact, may break correctness of the subclasses

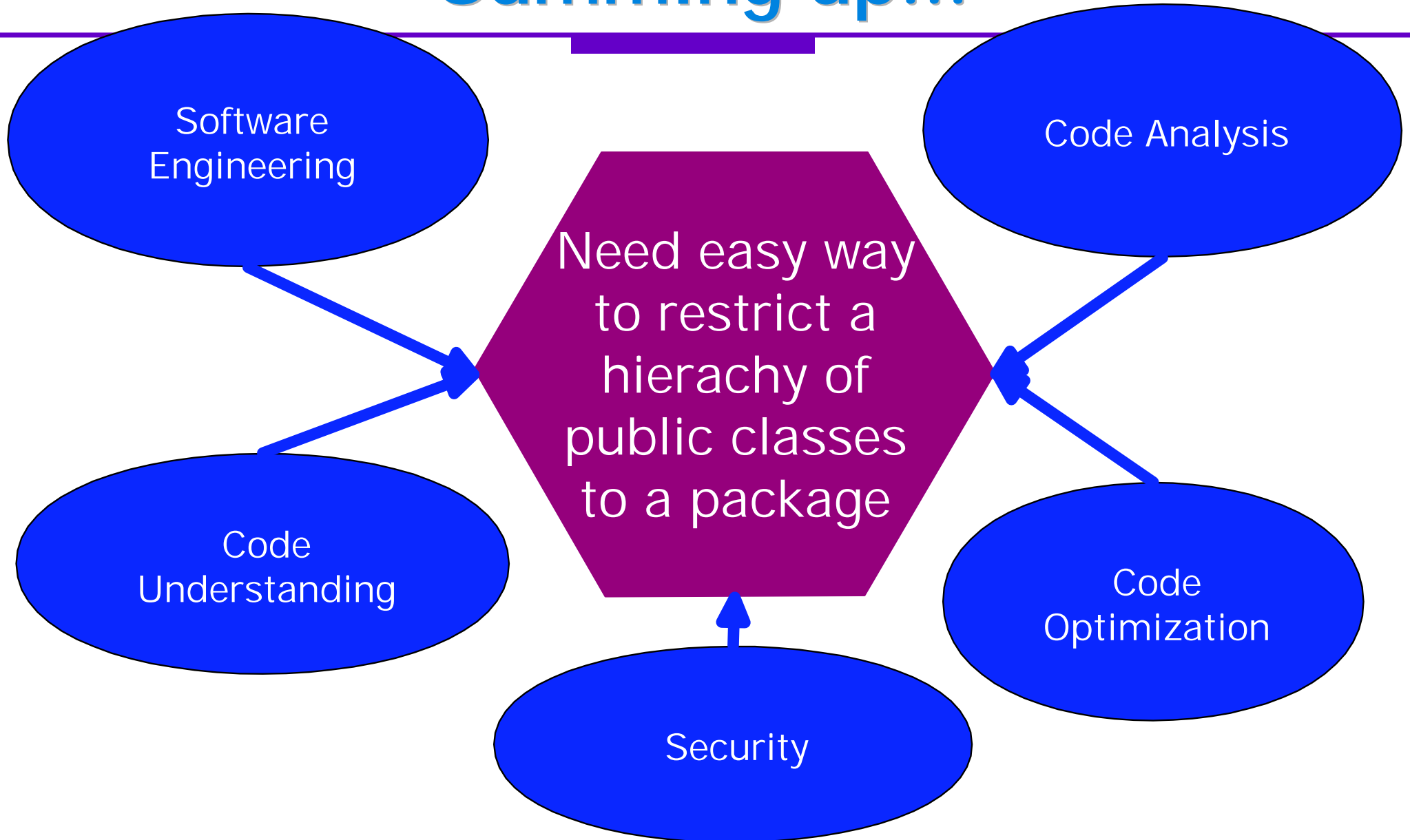
```
class Base{  
    protected int x;  
    protected void m(){  
        x++;  
    }  
    protected void n(){  
        x++; m();  
    }  
}
```

```
class Sub extends Base{  
    protected void m(){  
        n();  
    }  
}
```

Solutions to the Fragile Base Class Problem

- Pretend it's not there
- Use Mikhailov and Sekerinski's "disciplined inheritance"
 - Complex rules
- Maintain the complete hierarchy in the same maintenance unit
 - E.g., the same package

Summing up...



Sealed classes (at last...)

- Class C in package P is sealed if the hierarchy underneath C is bounded within P at processing time
- Can be implemented

- Via a new class modifier **sealed**

```
public sealed class Foo{...}
```

- By adding class sealing info to JAR Manifest

```
Manifest-Version: 1.0
```

```
Name: com/ibm/vehicle/Vehicle.class
```

```
Sealed: true
```

Experimentation

- Goal: does class sealing fit into *current programming practice*?
 - Based on Java SDK 1.2 rt.jar
- Only designer can designate a class as sealed!
- Experiment consists of:
 1. Find all classes in rt.jar that can be sealed
 2. Select a set of large and diverse applications
 3. Check how many 'sealed' classes in rt.jar are 'unsealed' by the applications