Introduction to Software Testing
Chapter 9.3
Integration and Object-Oriented Testing

Paul Ammann & Jeff Offutt

http://www.cs.gmu.edu/~offutt/softwaretest/

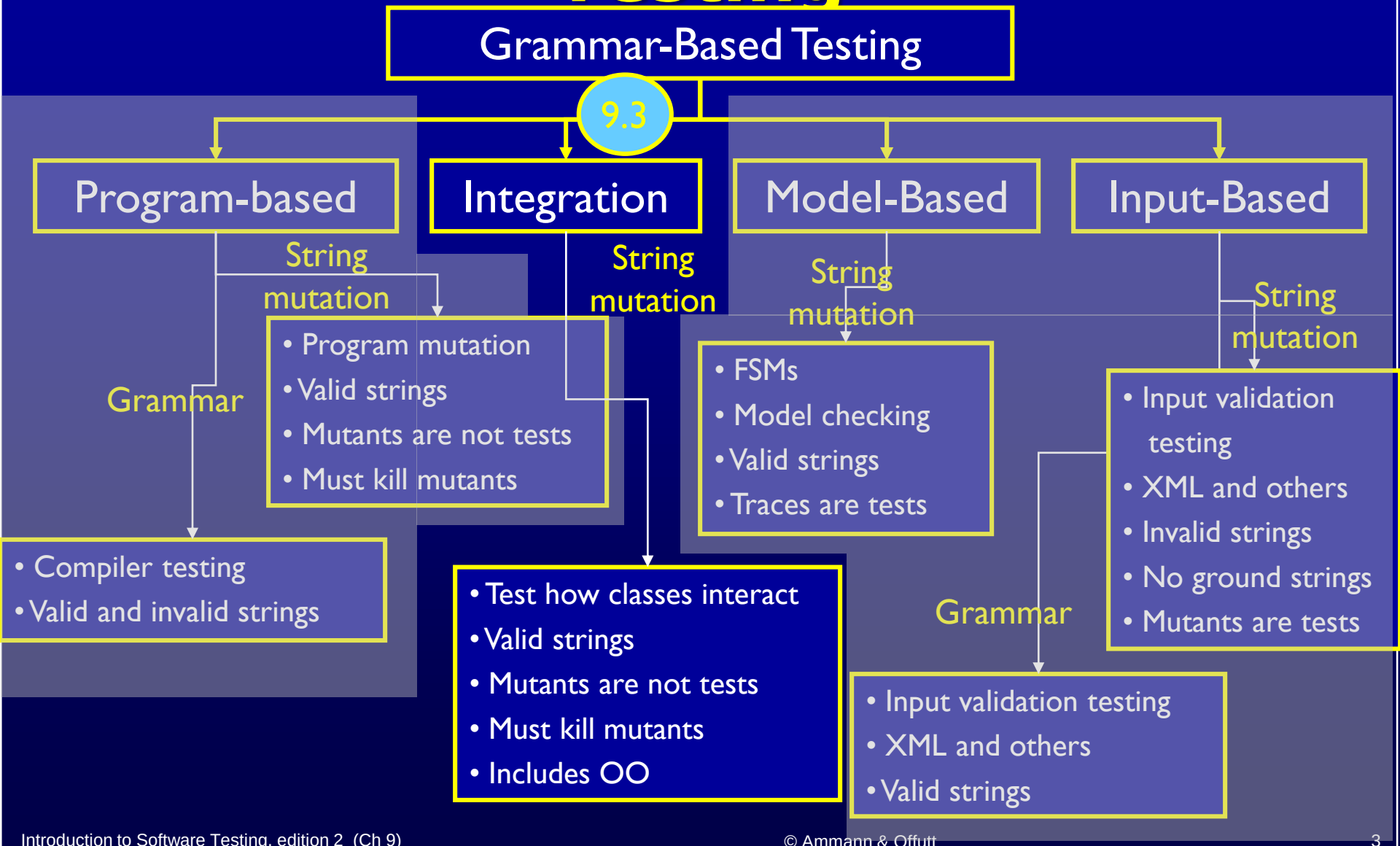
Integration and OO Testing

Integration Testing

Testing connections among separate program units

- In Java, testing the way classes, packages and components are connected
  - *"Component"* is used as a generic term
- This tests features that are unique to object-oriented programming languages
  - Inheritance, polymorphism and dynamic binding
- Integration testing is often based on couplings – the explicit and implicit relationships among software components

Instantiating Grammar-Based Testing



BNF Integration Testing (9.3.1)

There is no known use of grammar testing at the integration level

Integration Mutation (9.3.2)

- Faults related to component integration often depend on a mismatch of assumptions
  - Callee thought a list was sorted, caller did not
  - Callee thought all fields were initialized, caller only initialized some of the fields
  - Caller sent values in kilometers, callee thought they were miles

- Integration mutation focuses on mutating the connections between components
  - Sometimes called "*interface mutation*"
  - Both caller and callee methods are considered

Four Types of Mutation Operators

- Change a calling method by modifying values that are sent to a called method
- Change a calling method by modifying the call (overridden method)
- Change a called method by modifying values that enter and leave a method
  - Includes modifying parameters as well as variables from higher scopes (class level, package, public, etc.)
- Change a called method by modifying return statements from the method

Five Integration Mutation Operators

1. IPVR — Integration Parameter Variable Replacement

Each parameter in a method call is replaced by each other variable in the scope of the method call that is of compatible type

- This operator replaces primitive type variables as well as object.

Example

```
MyObject a, b;
. . .
callMethod (a);
Δ callMethod (b);
```

Five Integration Mutation Operators (2)

2. IUOI — Integration Unary Operator Insertion

Each expression in a method call is modified by inserting all possible unary operators in front and behind it

- The unary operators vary by language and type

Example

```
   callMethod (a);
Δ  callMethod (a++);
Δ  callMethod (++a);
Δ  callMethod (a--);
Δ  callMethod (--a);
```

Five Integration Mutation Operators (3)

3. IPEX — Integration Parameter Exchange

Each parameter in a method call is exchanged with each parameter of compatible types in that method call

- max (a, b) is mutated to max (b, a)

Example

```
Max (a, b);
Δ Max (b, a);
```

Five Integration Mutation Operators (4)

4. IMCD — Integration Method Call Deletion

Each method call is deleted. If the method returns a value and it is used in an expression, the method call is replaced with an appropriate constant value

- Method calls that return objects are replaced with calls to "new ()"

Example

```
  X = Max (a, b);
Δ X = new Integer (0);
```

Five Integration Mutation Operators (5)

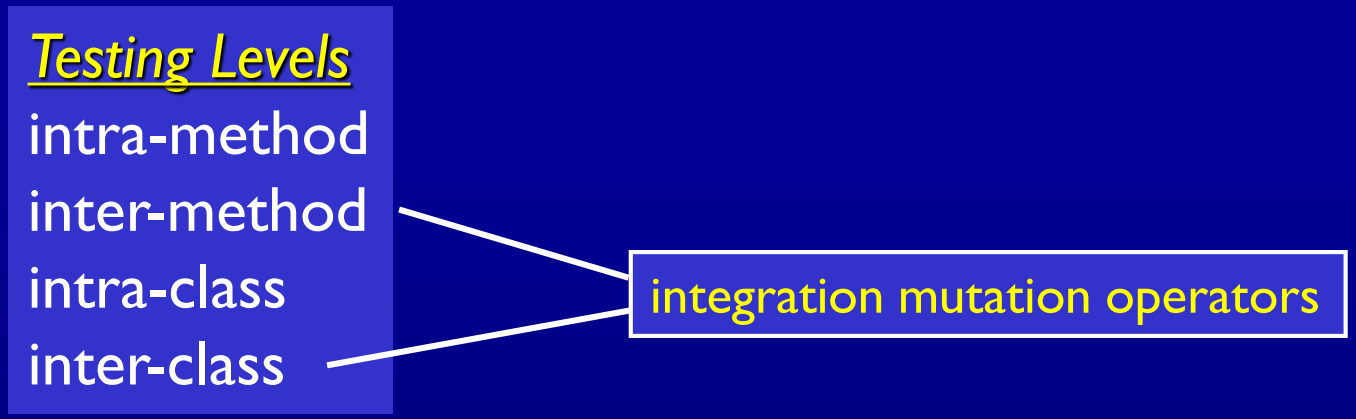
5. IREM — Integration Return Expression Modification

Each expression in each return statement in a method is modified by applying the UOI and AOR operators

Example

```
int myMethod ()
{
   return a + b;
Δ  return ++a + b;
Δ  return a - b;
}
```

Object-Oriented Mutation



- These five operators can be applied to non-OO languages
  - C, Pascal, Ada, Fortran, …
- They do not support object oriented features
  - Inheritance, polymorphism, dynamic binding
- Two other language features that are often lumped with OO features are information hiding (encapsulation) and overloading
- Even experienced programmers often get encapsulation and access control wrong

Encapsulation, Information Hiding and Access Control

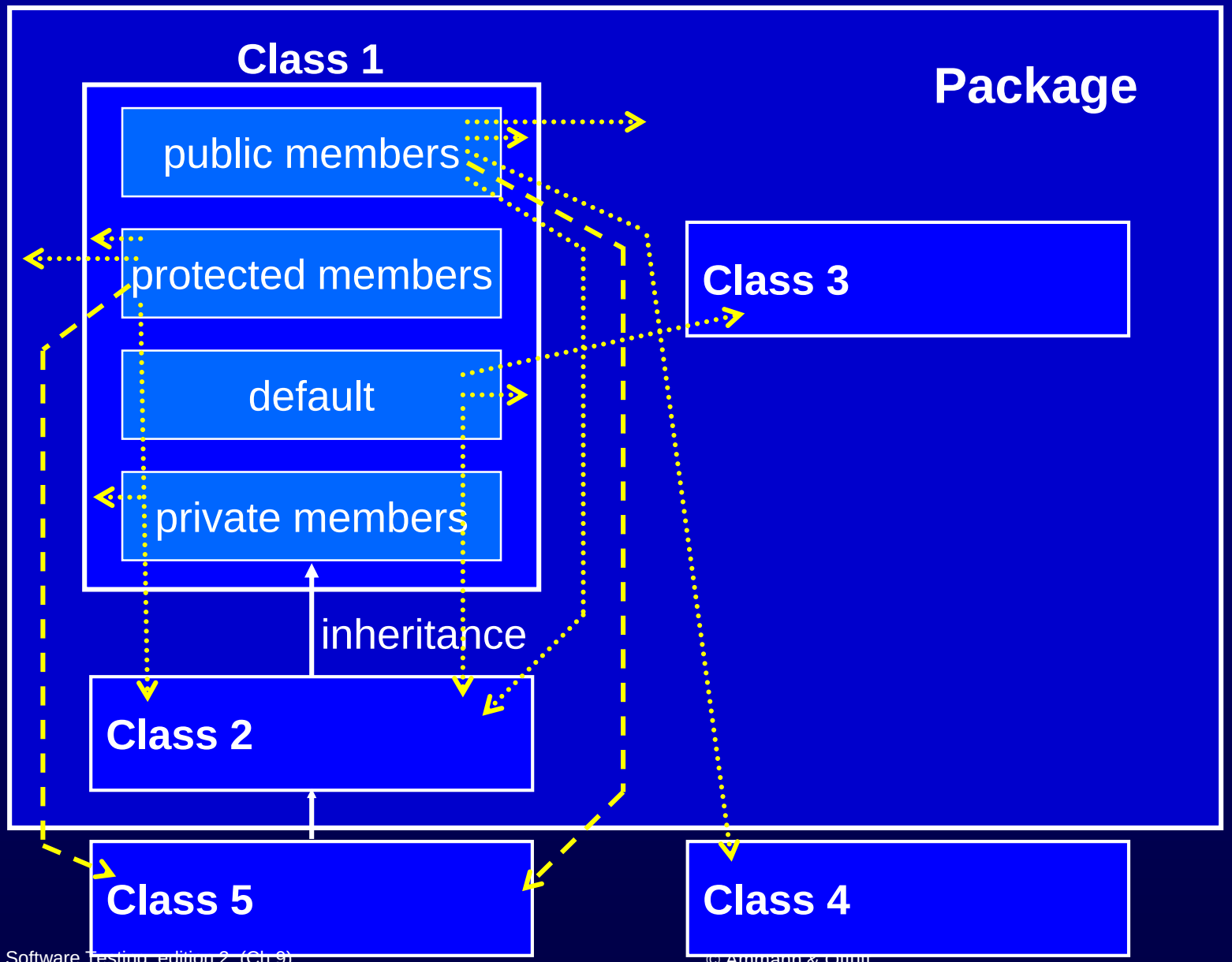
- *Encapsulation* : An abstraction mechanism to implement information hiding, which is a design technique that attempts to protect parts of the design from parts of the implementation
  - Objects can restrict access to their member variables and methods
- Java provides four access levels (C++ & C# are similar)
  - private
  - protected
  - public
  - default (also called package)
- Often not used correctly or understood, especially for programmers who are not well educated in design

Access Control in Java

Specifier	Same class	Same package	Different package subclass	Different package non-subclass
private	Y	n	n	n
package	Y	Y	n	n
protected	Y	Y	Y	n
public	Y	Y	Y	Y

- Most class variables should be private
- Public variables should seldom be used
- Protected variables are particularly dangerous – future programmers can accidentally override (by using the same name) or accidentally use (by mis-typing a similar name)
  - They should be called "unprotected"

Access Control in Java (2)



OO Language Features (Java)

- Method overriding

  Allows a method in a subclass to have the same name, arguments and result type as a method in its parent

- Variable hiding

  Achieved by defining a variable in a child class that has the same name and type of an inherited variable

- Class constructors

  Not inherited in the same way other methods are – must be explicitly called (using the *super* keyword)

- Each object has …
  - A *declared* type : *Parent P;*
  - An *actual* type : *P = new Child ();* or assignment : *P = Pold;*
  - Declared and actual types allow uses of the same name to reference different variables with different types

OO Language Feature Terms

- *Polymorphic attribute*
  - An object reference that can take on various types
  - Type the object reference takes on during execution can change
- *Polymorphic method*
  - Can accept parameters of different types because it has a parameter that is declared of type Object
- *Overloading*
  - Using the same name for different constructors or methods in the same class
- *Overriding*
  - A child class declares an object or method with a name that is already declared in an ancestor class
  - Easily confused with overloading because the two mechanisms have similar names and semantics
  - Overloading is in the same class, overriding is between a class and a descendant

More OO Language Feature Terms

- Members associated with a class (rather than with individual objects) are called class or static variables and methods
  - Static methods can operate only on static variables; not instance variables
  - Instance variables are declared at the class level and are available to objects
  - Class variables are declared with static
  - Local variables are declared within methods
- 25 ~~20~~ object-oriented mutation operators defined for Java – muJava
- Broken into 4 general categories

Class Mutation Operators for Java

(1) Encapsulation

AMC

(2) Inheritance

IHI, IHD, IOD, IOP, IOR, ISI, ISD, IPC

(3) Polymorphism

PNC, PMD, PPD, PCI, PCD, PCC, PRV, OMR, OMD, OAC

(4) Java-Specific

JTI, JTD, JSI, JSD, JID, JDC

OO Mutation Operators— *Encapsulation*

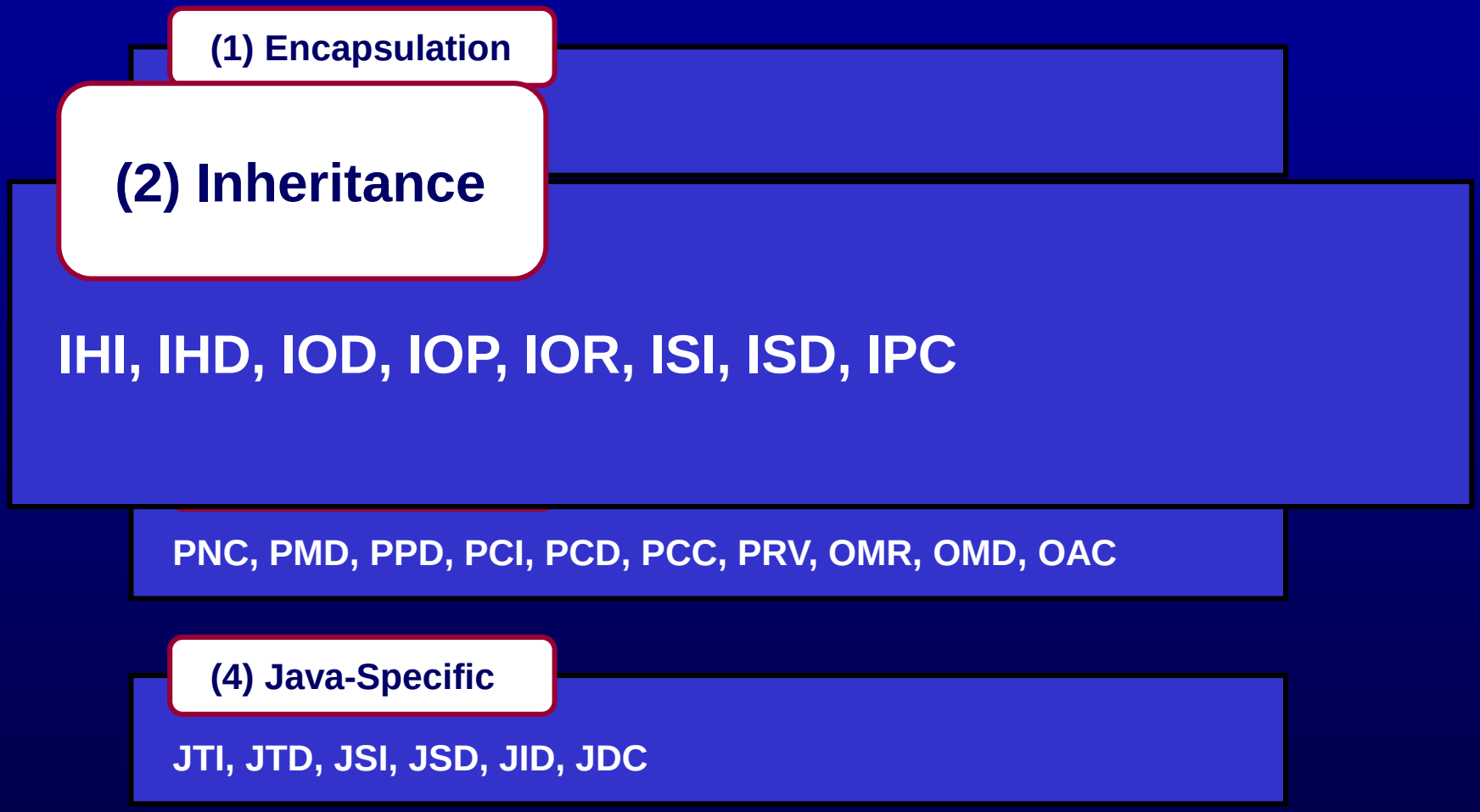
1. AMC — *Access Modifier Change*

The access level for each instance variable and method is changed to other access levels

Example

point
private int x; Δ1 public int x; Δ2 protected int x; Δ3 int x;

Class Mutation Operators for Java

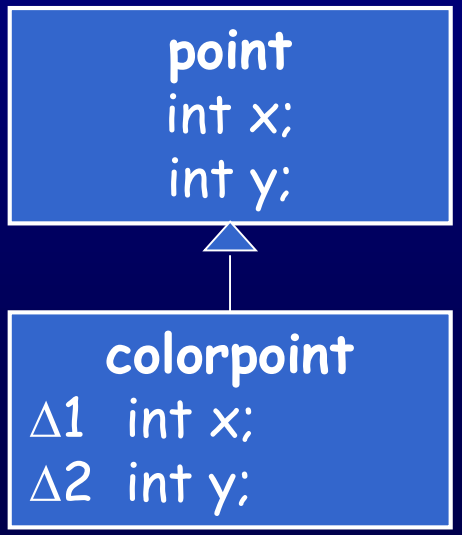


OO Mutation Operators— *Inheritance*

2. IHI — *Hiding Variable Insertion*

A declaration is added to hide the declaration of each variable declared in an ancestor

Example

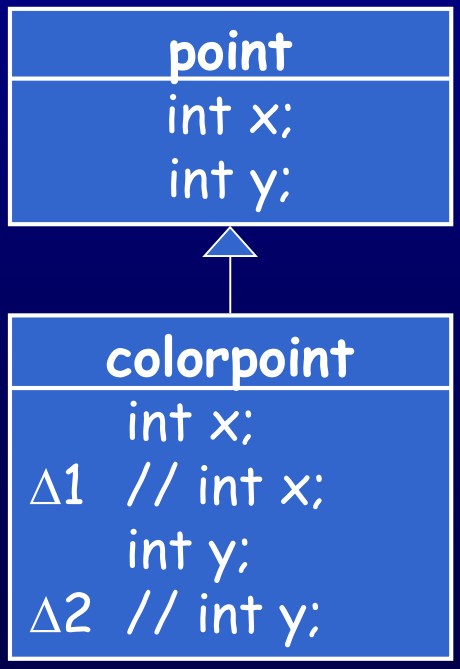


OO Mutation Operators— *Inheritance*

3. IHD — *Hiding Variable Deletion*

Each declaration of an overriding or hiding variable is deleted

Example

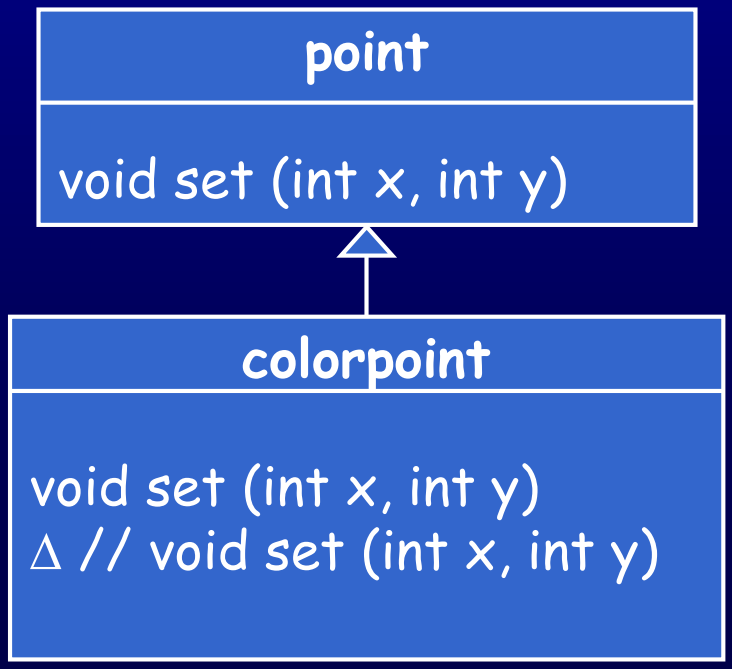


OO Mutation Operators—*Inheritance*

4. IOD — *Overriding Method Deletion*

Each entire declaration of an overriding method is deleted

Example

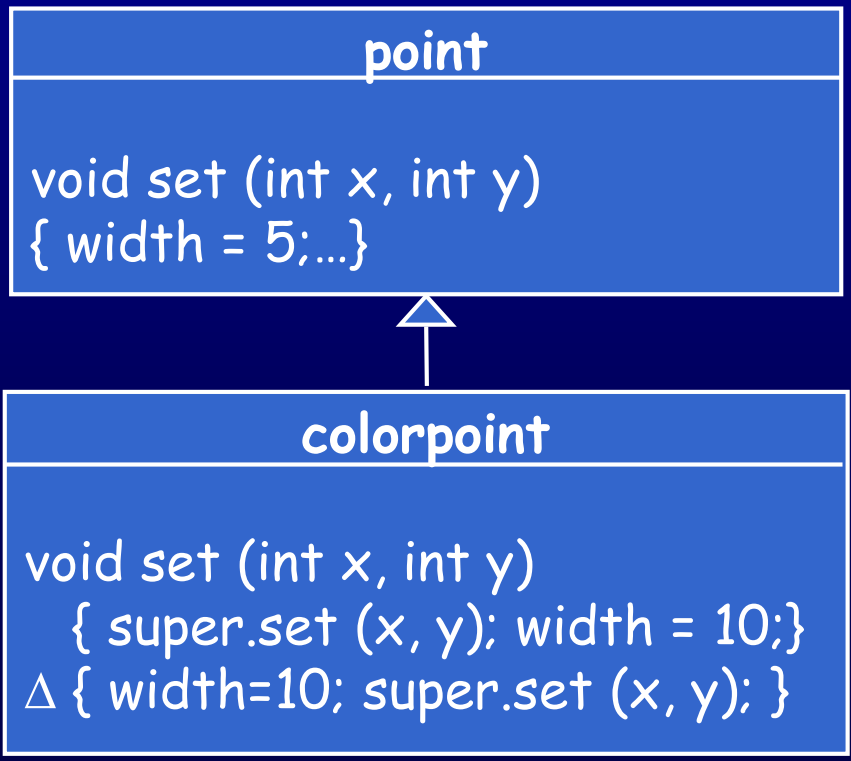


OO Mutation Operators—*Inheritance*

5. IOP — *Overridden Method Calling Position Change*

Each call to an overridden method is moved to the first and last statements of the method and up and down one statement

Example



Overriding methods in child classes often call the original method in the parent class, for example to modify a variable that is private to the parent.

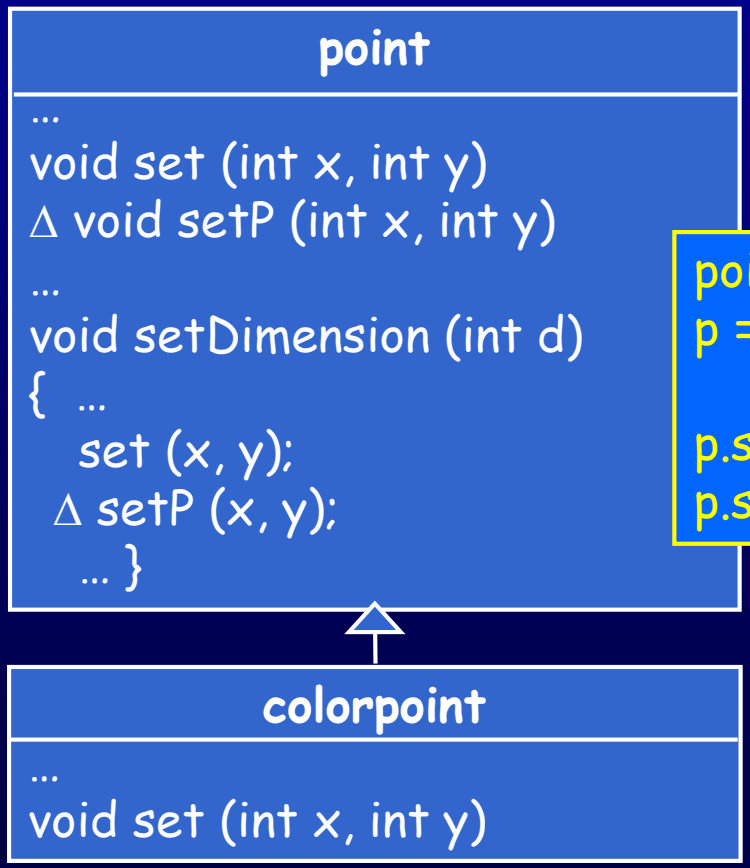
A common mistake to make is to call the parent's version at the wrong time, which can cause incorrect state behavior.

OO Mutation Operators—*Inheritance*

6. IOR — *Overridden Method Rename*

Renames the parent's versions of methods that are overridden in a subclass so that the overriding does not affect the parent's method

Example




```
point p;
p = new colorpoint ();
      ...
p.set (1, 2);
p.setDimension (3);
```

The IOR operator is designed to check whether an overriding method causes problems with other methods

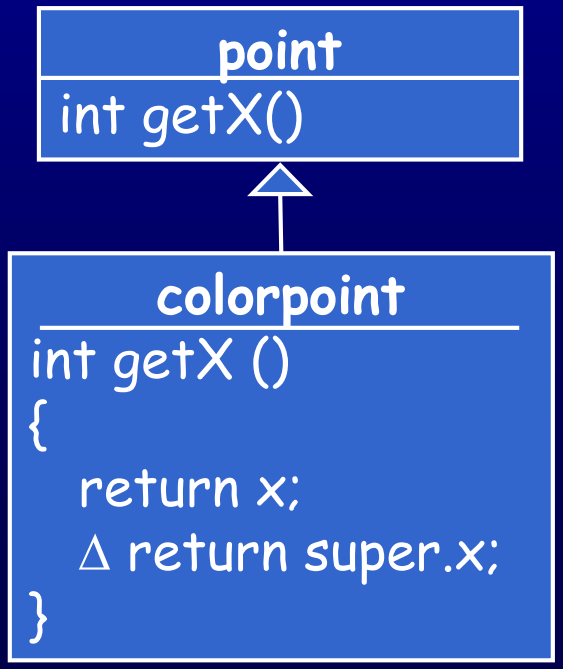
in p.setDimension(), it shall call colorpoint's version of set(). In this case, colorpoint's version of setDimension() may have an interaction with the parent's set() version that can has unintended consequence

OO Mutation Operators—*Inheritance*

7. ISI — *Super Keyword Insertion*

Inserts the super keyword before overriding variables or methods (if the name is also defined in an ancestor class)

Example

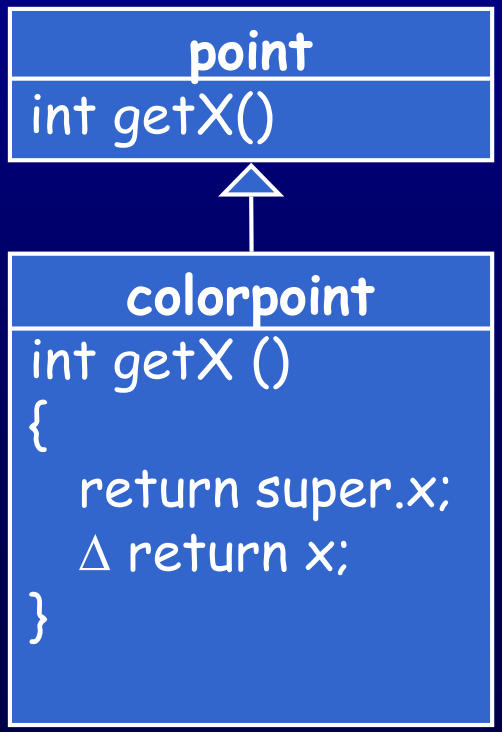


OO Mutation Operators—*Inheritance*

8. ISD — *Super Keyword Deletion*

Delete each occurrence of the super keyword

Example

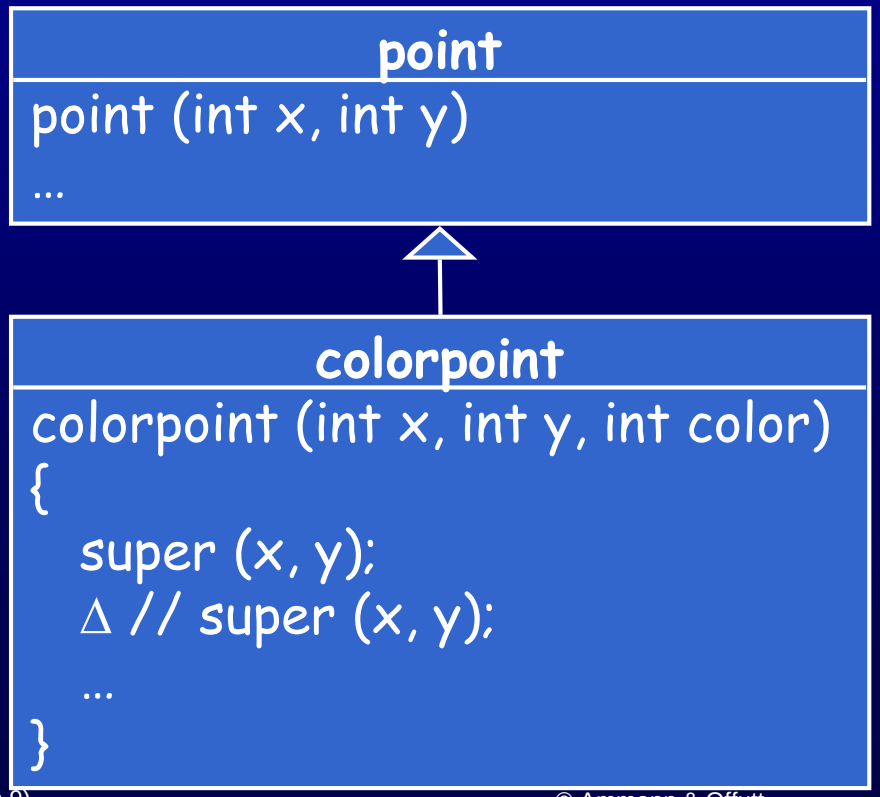


OO Mutation Operators—*Inheritance*

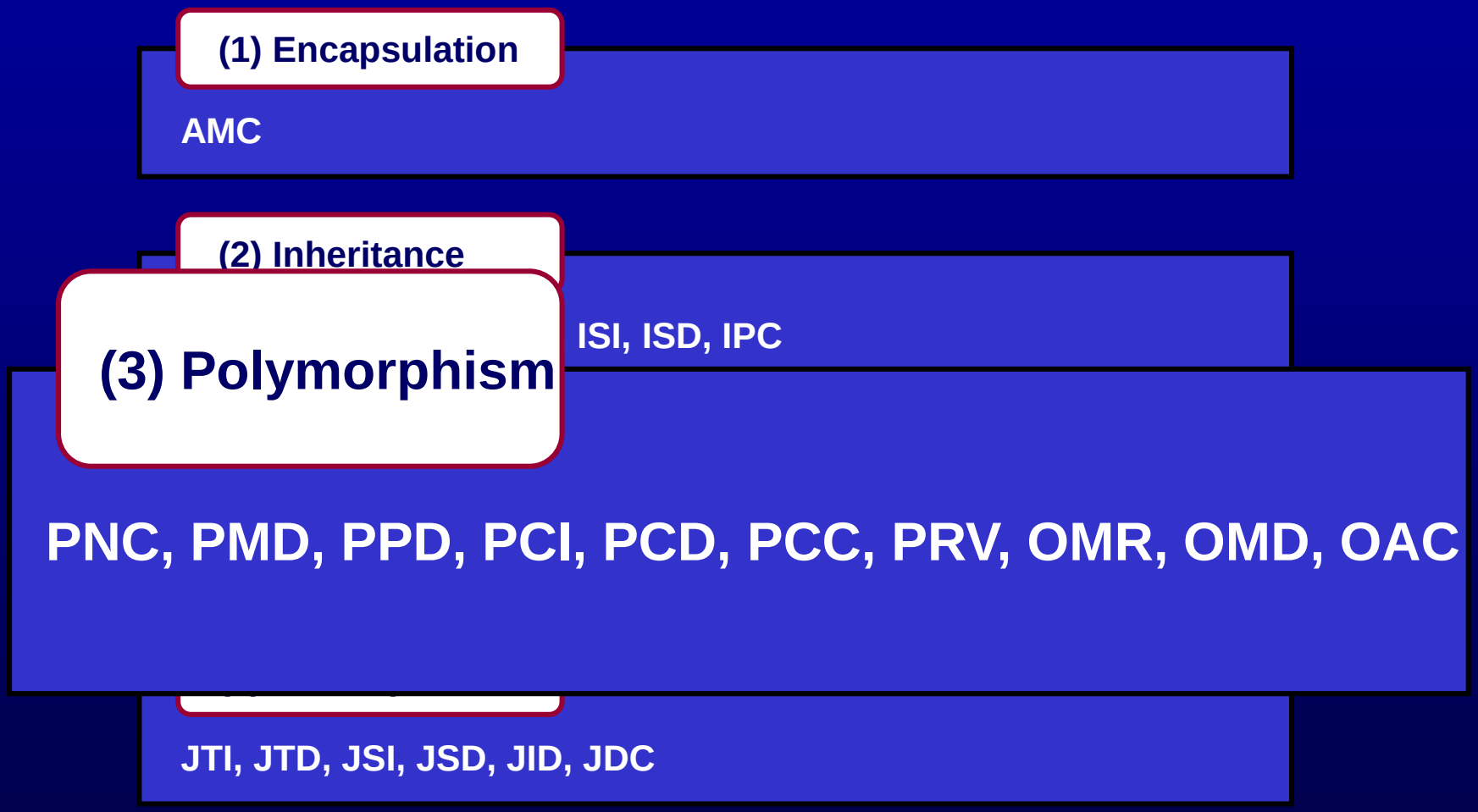
9. IPC — *Explicit Parent Constructor Deletion*

Each call to a super constructor is deleted

Example



Class Mutation Operators for Java

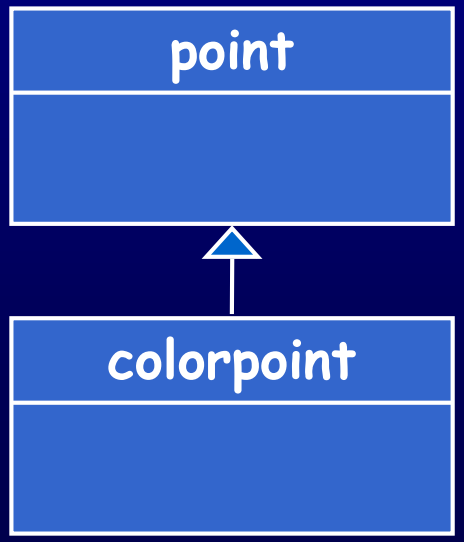


OO Mutation Operators— *Polymorphism*

10. PNC — *new Method Call With Child Class Type*

The actual type of a new object is changed in the new() statement

Example



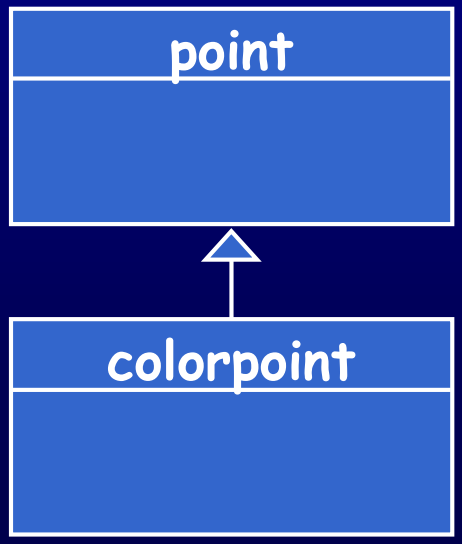

```
  point p;
  p = new point ();
Δ p = new colorpoint ();
```

OO Mutation Operators— *Polymorphism*

11. PMD — *Member Variable Declaration with Parent Class Type*

The declared type of each new object is changed in the declaration

Example



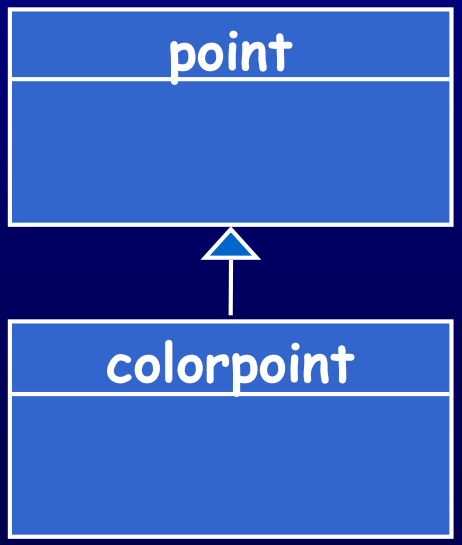

```
  point p;
Δ colorpoint p;
  p = new colorpoint ();
```

OO Mutation Operators— *Polymorphism*

12. PPD — *Parameter Variable Declaration with Child Class Type*

The declared type of each parameter object is changed in the declaration

Example



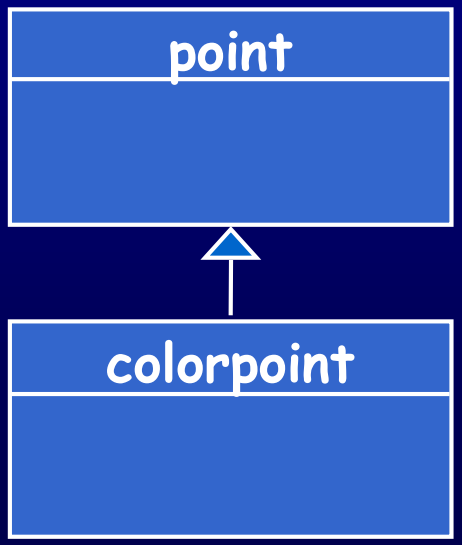

```
   boolean equals (point p)
   { . . . }
Δ  boolean equals (colorpoint p)
   { . . .}
```

OO Mutation Operators— *Polymorphism*

13. PCI — *Type Cast Operator Insertion*

The actual type of an object reference is changed to the parent or to the child of the original declared type

Example



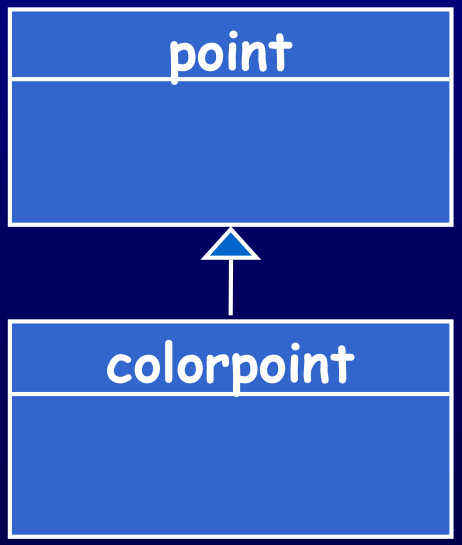

```
   point p;
   p = new colorpoint ();
   int x = p.getX ();
Δ  int x = ((point) p).getX ();
```

OO Mutation Operators— *Polymorphism*

14. PCD — *Type Cast Operator Deletion*

Type casting operators are deleted

Example



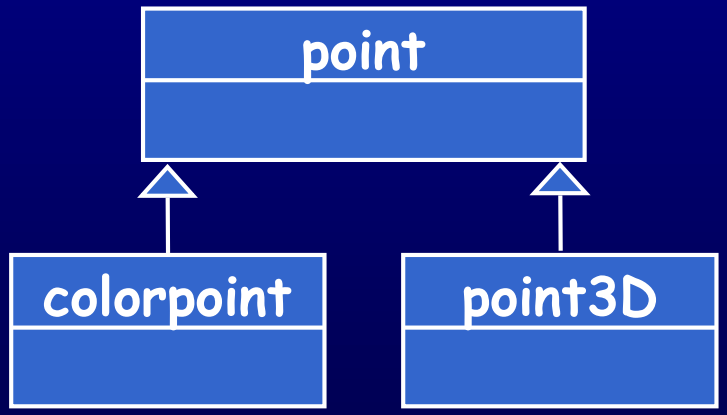

```
   point p;
   p = new colorpoint ();
   int x = ((point) p).getX ();
Δ  int x = p.getX ();
```

OO Mutation Operators— *Polymorphism*

15. PPC — *Cast Type Change*

Changes the type to which an object reference is being cast

Example



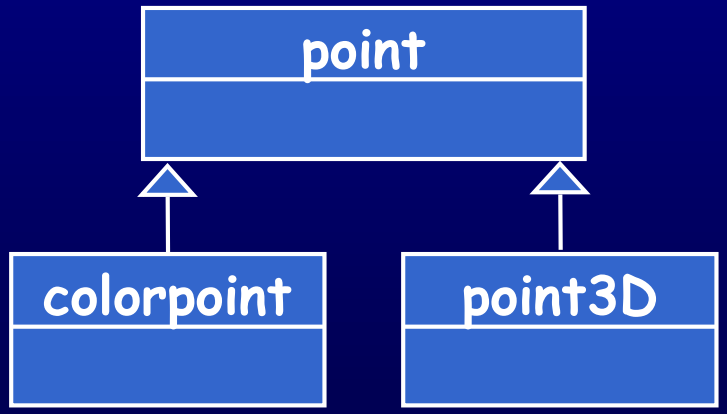

```
   point p = new point (0, 0);
   int x = ((colorpoint) p).getX ();
Δ  int x = ((point3D) p).getX ();
```

OO Mutation Operators— *Polymorphism*

16. PRV — *Reference Assignment with Other Compatible Type*

The right side objects of assignment statements are changed to refer to objects of a compatible type

Example




```
point p;
colorpoint cp = new colorpoint (0, 0);
point3D p3d = new point3D (0, 0, 0);
p = cp;
Δ p = p3d;
```

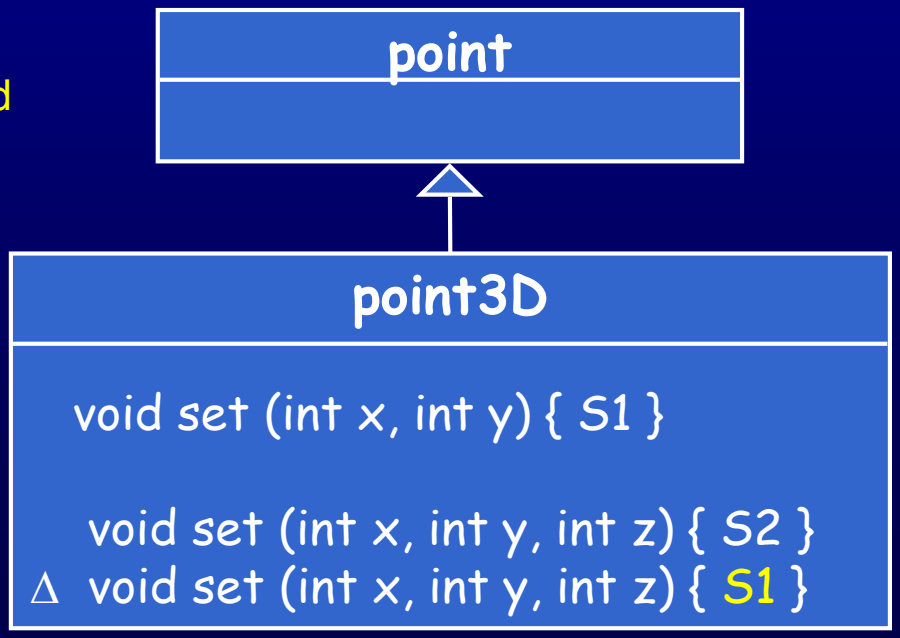
OO Mutation Operators— *Polymorphism*

17. OMR — *Overloading Method Contents Replace*

For each pair of methods that have the same name, the bodies are interchanged

The OMR operator is designed to check that overloaded methods are invoked appropriately

Example



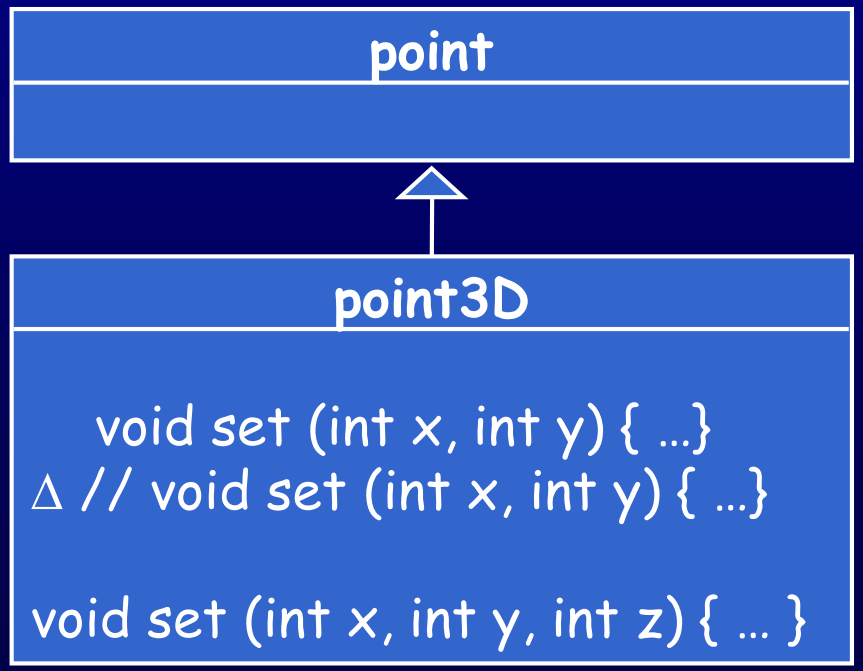
OO Mutation Operators— *Polymorphism*

18. OMD — *Overloading Method Deletion*

Each overloaded method declaration is deleted, one at a time

The OMD operator ensures coverage of overloaded methods; that is, all the overloaded methods must be invoked at least once.

Example



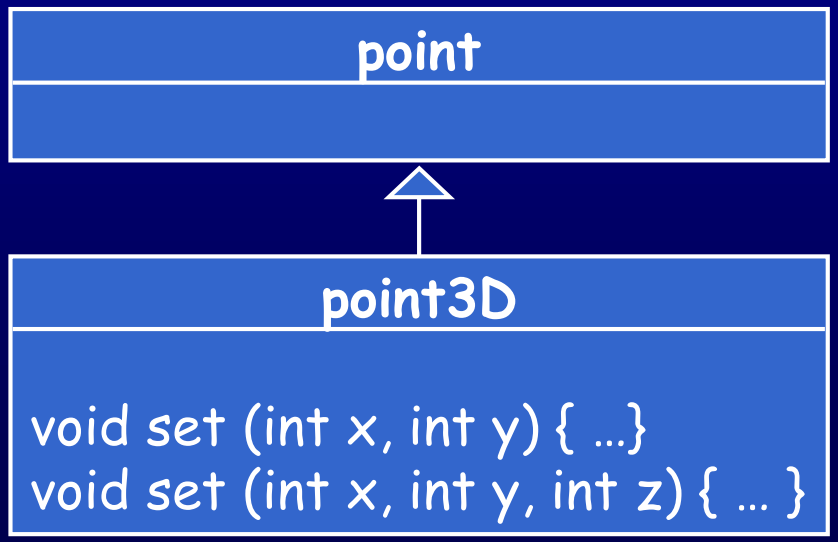
The OMD operator deletes overloading method declarations, one at a time in turn. If the mutant still works correctly without the deleted method, there may be an error in invoking one of the overloading methods; the incorrect method may be invoked or an incorrect parameter type conversion has occurred

OO Mutation Operators— *Polymorphism*

19. OAC — *Arguments of Overloading Method Call Change*

The order of the arguments in method invocations is changed to be the same as that of another overloading method, if one exists

Example




```
   point p = new point3D ();
   p.set (5, 7, 9);
Δ  p.set (5, 7);
```

the OAC operator causes a different method to be called, thus checking for a common fault in the use of overloading

Class Mutation Operators for Java

(1) Encapsulation

AMC

(2) Inheritance

IHI, IHD, IOD, IOP, IOR, ISI, ISD, IPC

(3) Polymorphism

PCD, PCC, PRV, OMR, OMD, OAC

(4) Java-Specific

JTI, JTD, JSI, JSD, JID, JDC

OO Mutation Operators—*Language Specific*

20. JTI — *this Keyword Insertion*

The keyword this is inserted whenever possible

Example

point

```
...
void set (int x, int y)
{
     x = x;
Δ1 this.x = x;
     y = y;
Δ2 this.y = y;
}
...
```

The JTD operator checks if the member variables are used correctly if they are hidden by a method parameters by replacing occurrences of "this.x" with "x" when "x" is both a parameter and an instance variable.

OO Mutation Operators—*Language Specific*

21. JTD — *this Keyword Deletion*

The keyword this is deleted whenever possible

Example

point

```
...
void set (int x, int y)
{
    this.x = x;
Δ1 x = x;
    this.y = y;
Δ2 y = y;
}
...
```

OO Mutation Operators—*Language Specific*

22. JSI — *Static Modifier Insertion*

The `static` modifier is added to instance variables

Example

point

```
    public int x = 0;
Δ1  public static int x = 0;
    public int Y = 0;
Δ2  public static int y = 0;

    ...
```

The JSI is designed to validate behavior of instance and class variables

OO Mutation Operators—*Language Specific*

23. JSD — *Static Modifier Deletion*

Each instance of the `static` modifier is removed

Example

point

```
    public static int x = 0;
Δ1  public int x = 0;
    public static int Y = 0;
Δ2  public int y = 0;

    ...
```

OO Mutation Operators—*Language Specific*

24. JID — *Member Variable Initialization Deletion*

Remove initialization of each member variable

Example

point

```
   int x = 5;
Δ  int x;
   ...
```

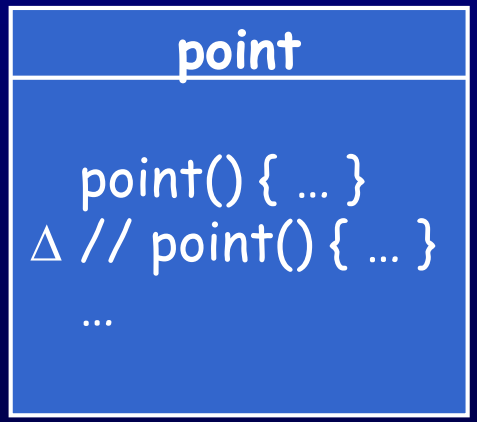
The JID operator removes the initialization of member variables in the variable declaration so that member variables are initialized to the appropriate default values of Java. This is designed to ensure correct initializations of instance variables

OO Mutation Operators—*Language Specific*

25. JDC — *Java-supported Default Constructor Deletion*

Delete each declaration of default constructor (with no parameters)

Example



The JDC operator forces Java to create a default constructor by deleting the implemented default constructor. It is designed to check if the user-defined default constructor is implemented properly

OO Mutation Operators—*Language Specific*

26. EOA — *Reference Assignment and Content Assignment Replacement*

Replaces an assignment of a pointer reference with a copy of the object using clone()

Example

Mutant class

```
Stack s1, s2;
s1 = new Stack();
s2 = s1;
Δ // s2 = s1.clone();
   ...
```

Object references in Java are always through pointers. Although pointers in Java are typed, which is considered to help prevent certain types of faults, there are still mistakes that programmers can make. One common mistake is that of using an object reference instead of the contents of the object the pointer references

OO Mutation Operators—*Language Specific*

27. EOC — *Reference Comparison and Content Comparison Replacement*

Replaces the comparison of the contents of objects with an equal() method

Example

Mutant class

```
Integer i1 = new Integer (7);
Integer i2 = new Integer (7);
boolean b = (i1==i2);
Δ //  boolean b = (i1.equals (i2));
   ...
```

The EOC operator considers another common mistake with objects and object references. Comparisons of object references check whether the two references point to the same data object in memory. The EOC operator targets faults programmers can easily make when confusing the reference of an object and its state

OO Mutation Operators—*Language Specific*

28. EAM — *Accessor Method Change*

Change an accessor method name with other compatible accessor method names, where *compatible* means that the signatures are the same

Example

Mutant class

```
point.getX();
Δ //  point.getY();
   ...
```

This type of mistake occurs because classes with multiple instance variables may wind up having many accessor methods with the same signature and very similar names. As a result, programmers easily get them confused. To kill this mutant a test case will have to produce incorrect output as a result of calling the wrong method

OO Mutation Operators—*Language Specific*

29. EMM — *Modifier Method Change*

Change a modifier method name with other compatible modifier method names, where *compatible* means that the signatures are the same

Example

Mutant class

```
point.setX(2);
Δ //  point.setY(2);
   ...
```

Class Mutation Operators for Java

(1) Encapsulation

AMC

(2) Inheritance

IHI, IHD, IOD, IOP, IOR, ISI, ISD, IPC

(3) Polymorphism

PNC, PMD, PPD, PCI, PCD, PCC, PRV, OMR, OMD, OAC

(4) Java-Specific

JTI, JTD, JSI, JSD, JID, JDC

Integration Mutation Summary

- Integration testing often looks at couplings
- We have not used grammar testing at the integration level
- Mutation testing modifies callers and callees
- OO mutation focuses on inheritance, polymorphism, dynamic binding, information hiding and overloading
  - The access levels make it easy to make mistakes in OO software
- muJava is an educational & research tool for mutation testing of Java programs
  - http://cs.gmu.edu/~offutt/mujava/