# 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

# 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
- Change a called method by modifying values that enter and leave a method
  - Includes 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)

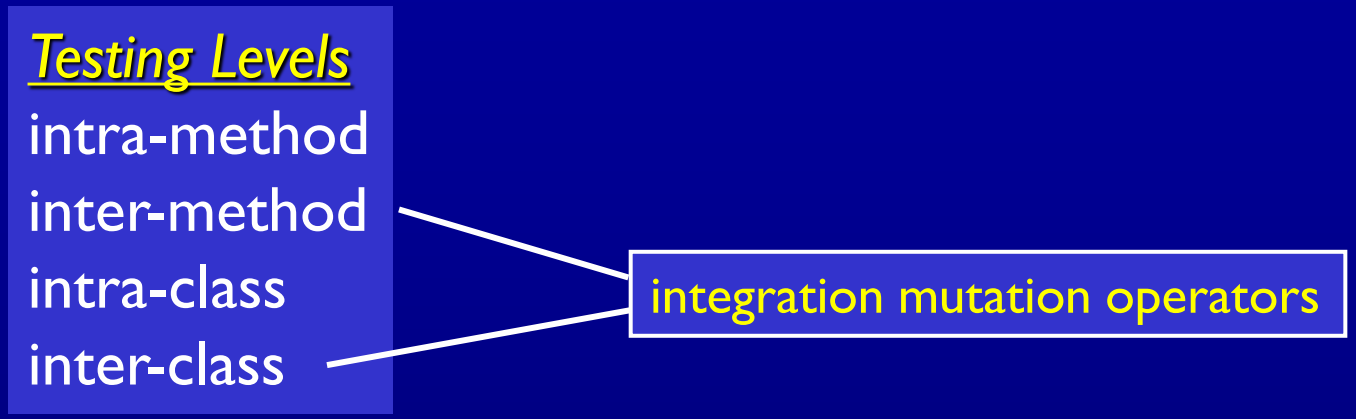
*5. IREM — Integration Return Expression Modification*

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"

# 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

- 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 are called class or instance 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
- 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*

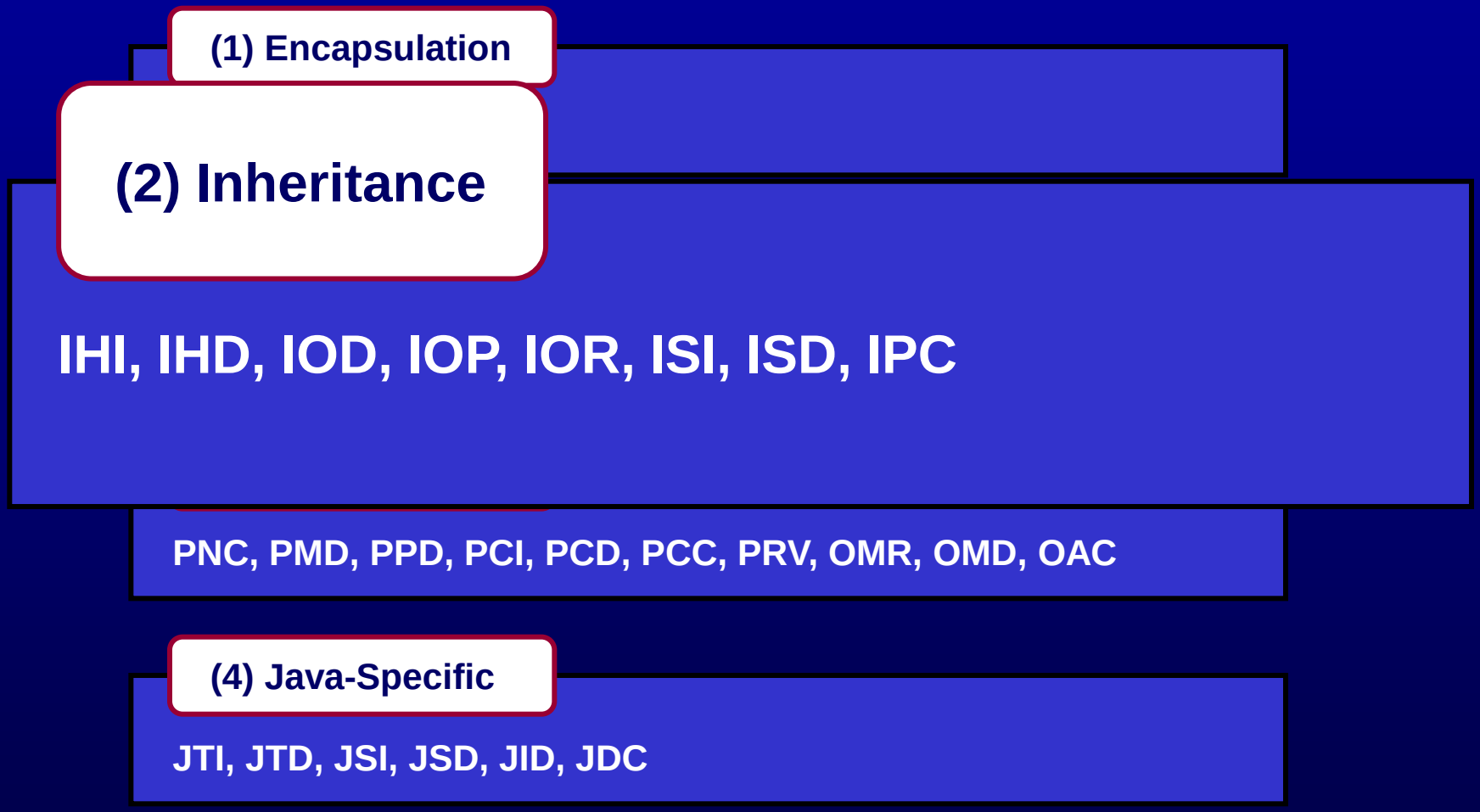
## 1. AMC — *Access Modifier Change*

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

***Example***

| point |
| --- |
| private int x;<br>Δ1 public int x;<br>Δ2 protected int x;<br>Δ3 int x; |

# Class Mutation Operators for Java

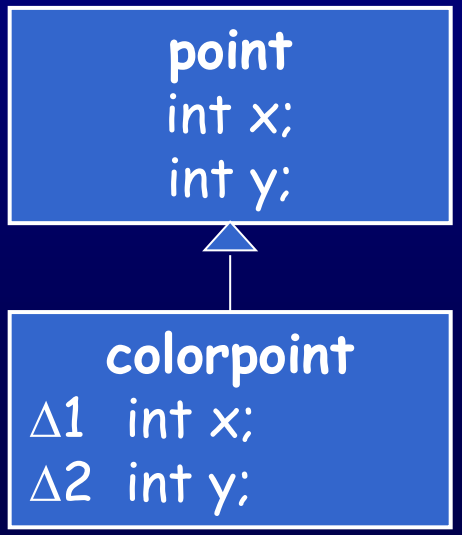


# OO Mutation Operators—*Inheritance*

## 2. IVI — *Hiding Variable Insertion*

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

***Example***

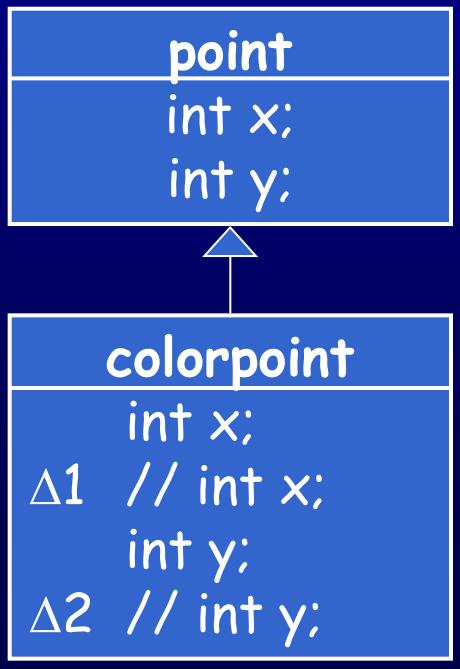


# OO Mutation Operators— *Inheritance*

## 3. IVD — *Hiding Variable Deletion*

Each declaration of an overriding or hiding variable is deleted

***Example***

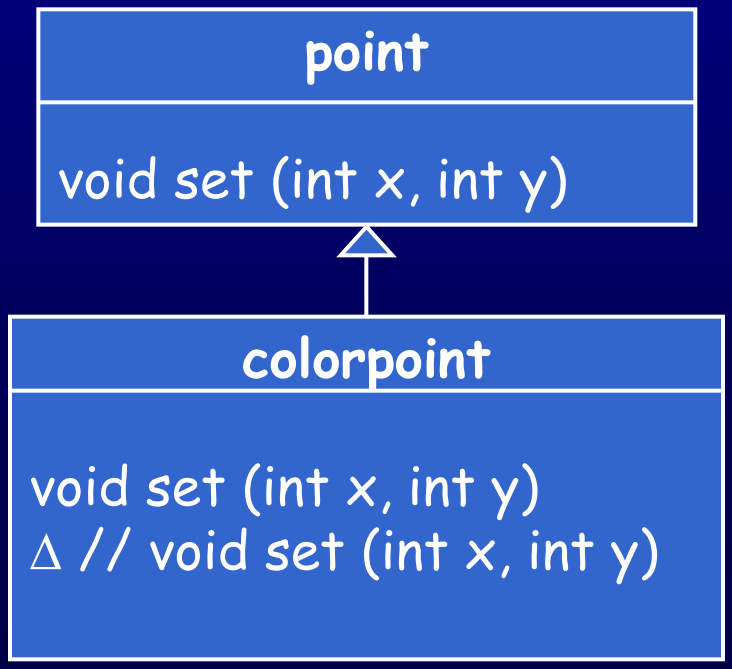


# OO Mutation Operators—*Inheritance*

## 4. IOD — *Overriding Method Deletion*

Each entire declaration of an overriding method is deleted

***Example***

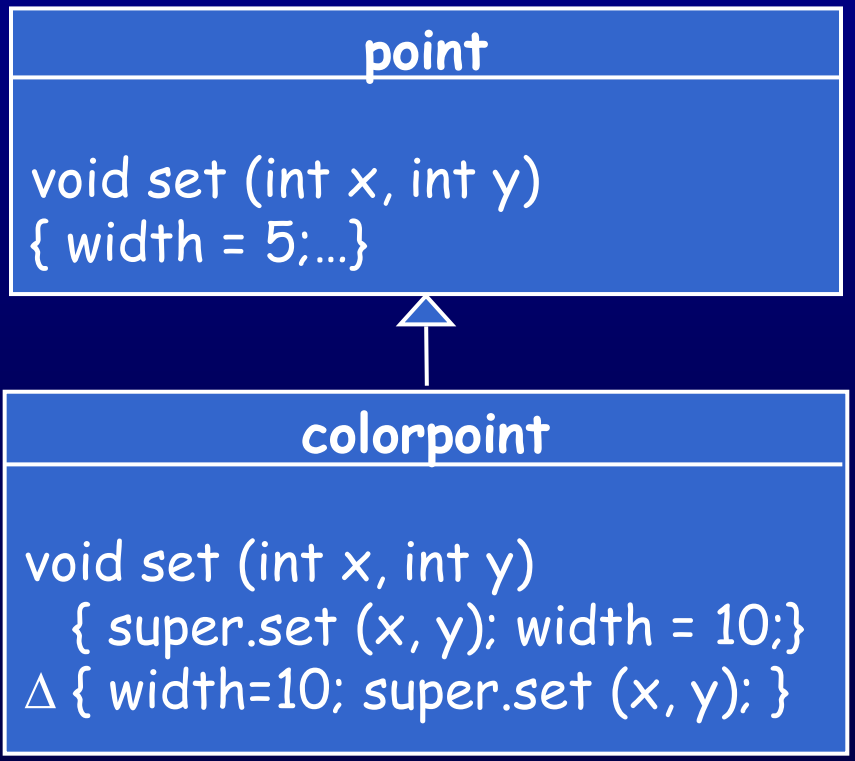


# OO Mutation Operators—*Inheritance*

## 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***

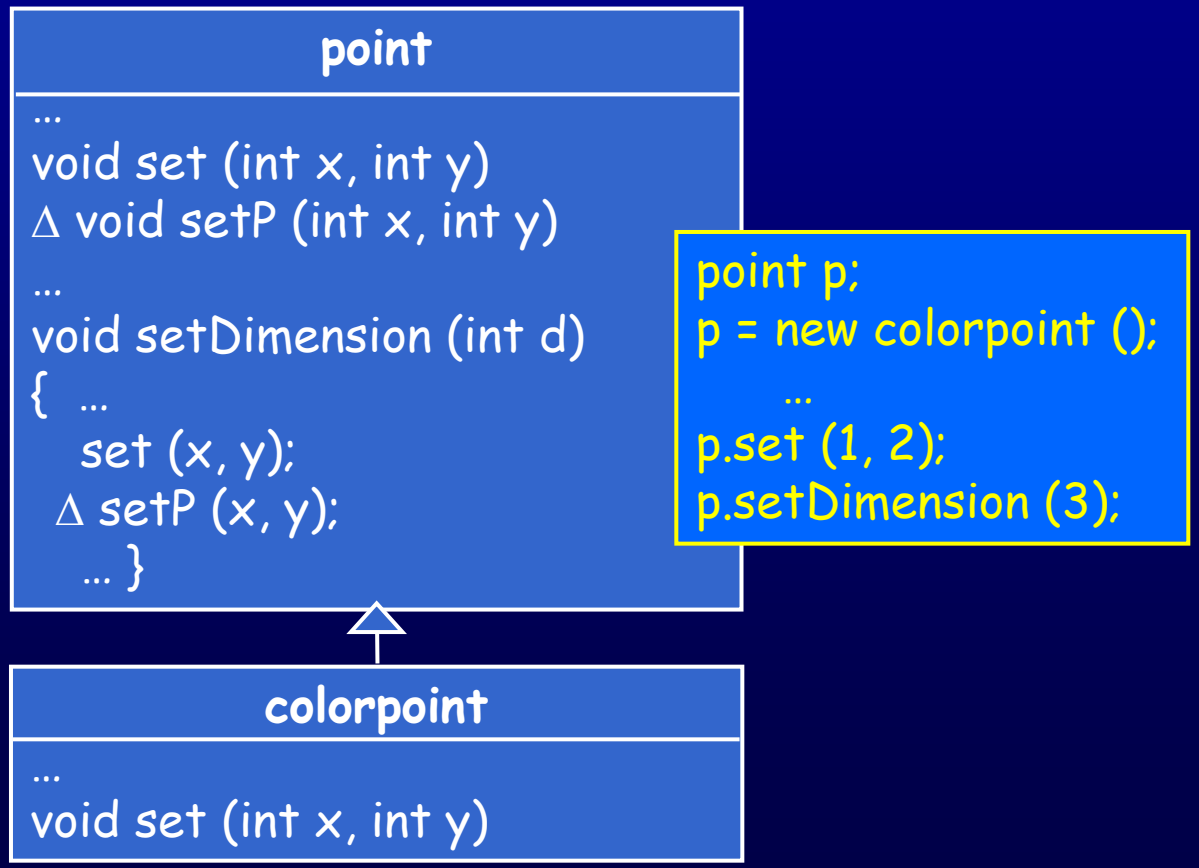


# OO Mutation Operators—*Inheritance*

## 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***

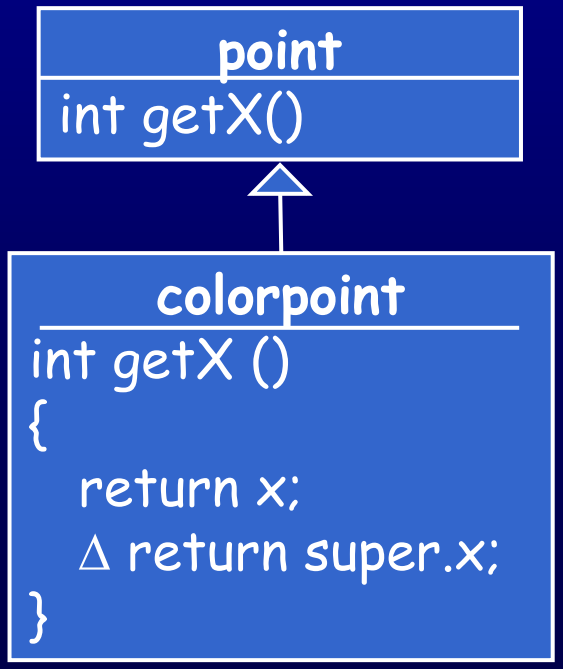


# OO Mutation Operators—*Inheritance*

## 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***

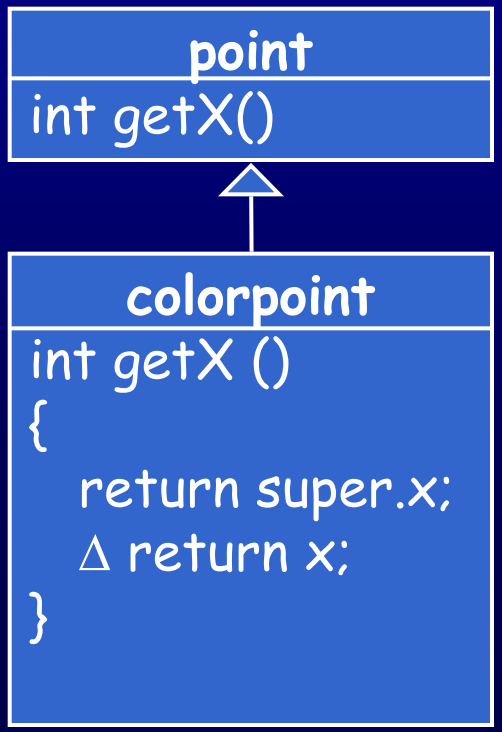


# OO Mutation Operators—*Inheritance*

## 8. ISD — *Super Keyword Deletion*

Delete each occurrence of the super keyword

***Example***

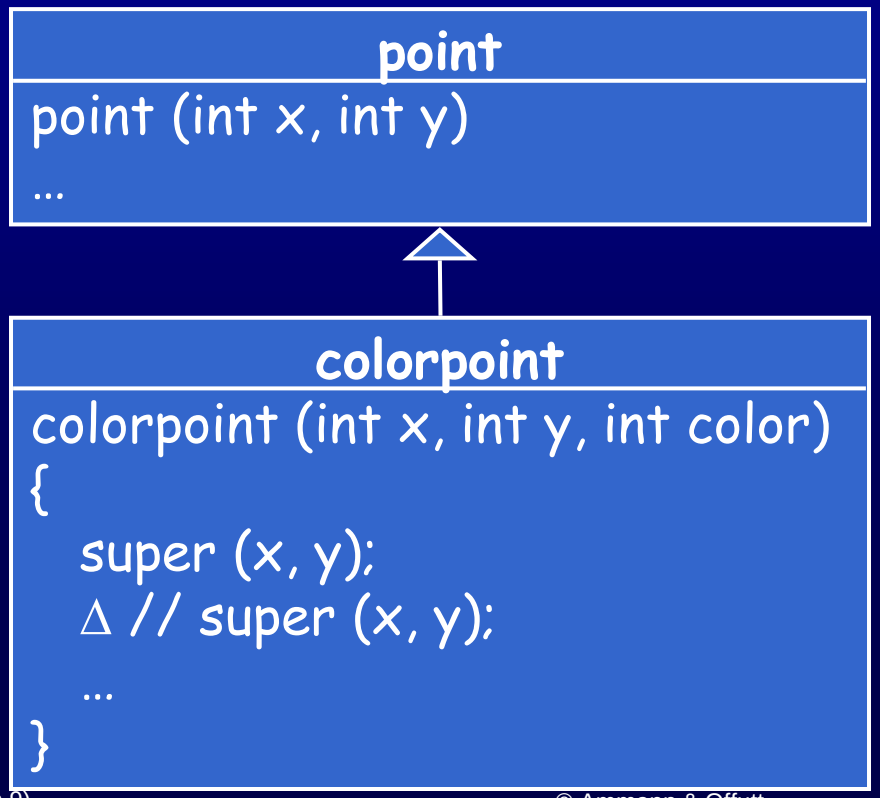


# OO Mutation Operators—*Inheritance*

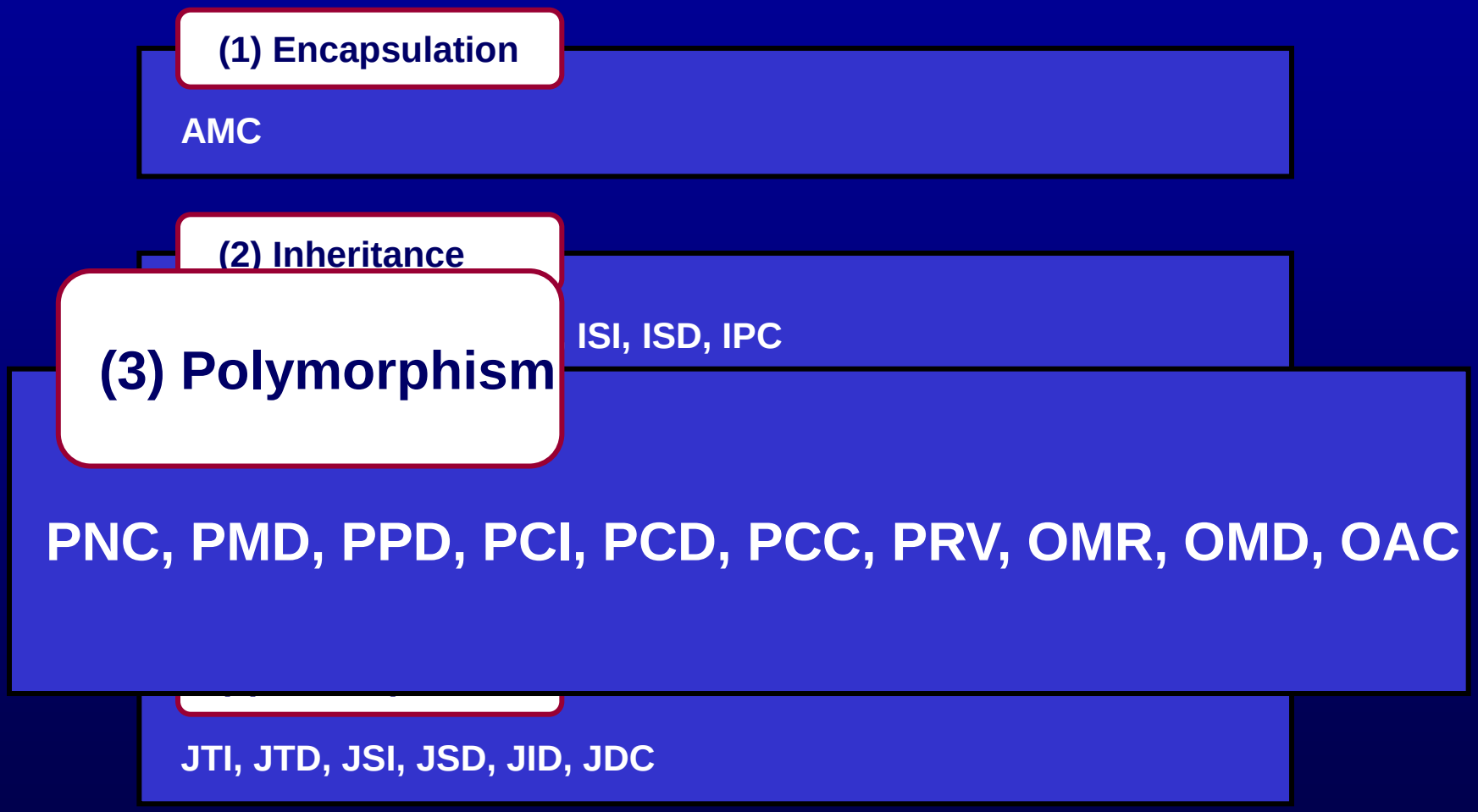
## 9. IPC — *Explicit Parent Constructor Deletion*

Each call to a **super** constructor is deleted

***Example***



# Class Mutation Operators for Java

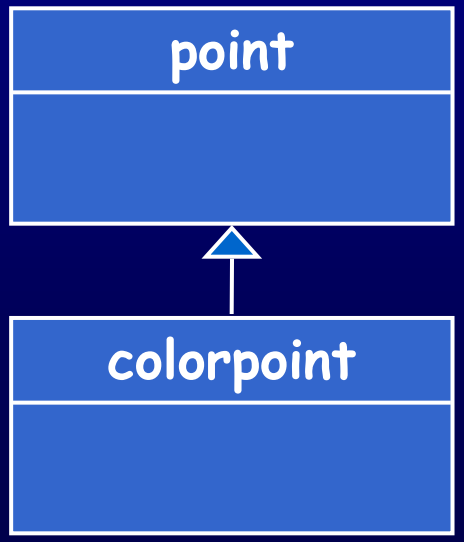


# OO Mutation Operators— *Polymorphism*

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

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

***Example***



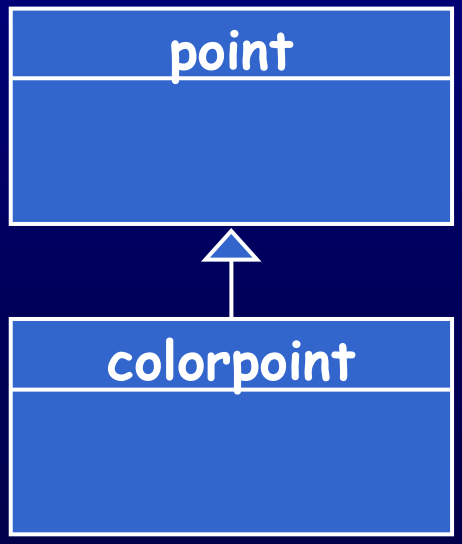

```
   point p;
   p = new point ();
Δ  p = new colorpoint ();
```

# 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***



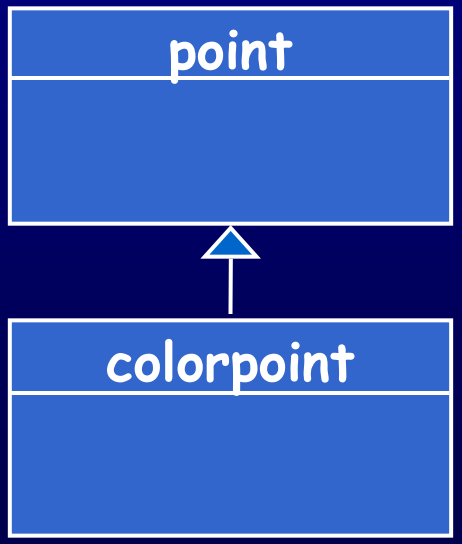

```
   point p;
Δ colorpoint p;
   p = new colorpoint ();
```

# 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***



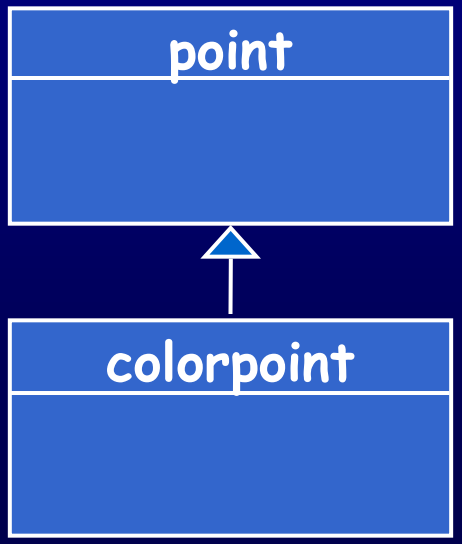

```
   boolean equals (point p)
   { . . . }
Δ  boolean equals (colorpoint p)
   { . . .}
```

# 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***



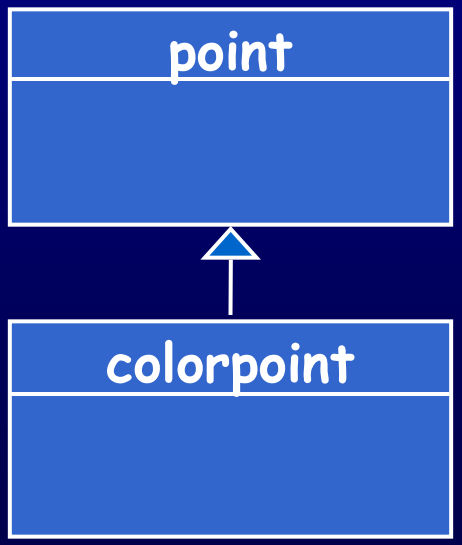

```
   point p;
   p = new colorpoint ();
   int x = p.getX ();
Δ  int x = (point) p.getX ();
```

# OO Mutation Operators— *Polymorphism*

## 14. PCD — *Type Cast Operator Deletion*

Type casting operators are deleted

***Example***



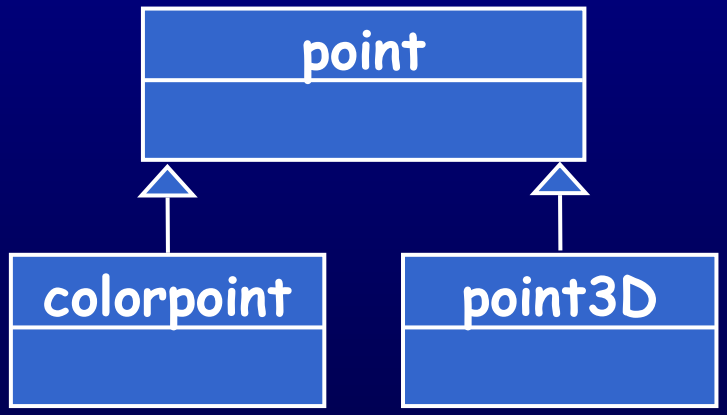

```
   point p;
   p = new colorpoint ();
   int x = (point) p.getX ();
Δ  int x = p.getX ();
```

# OO Mutation Operators—*Polymorphism*

## 15. PPC — *Cast Type Change*

Changes the type to which an object reference is being cast

***Example***



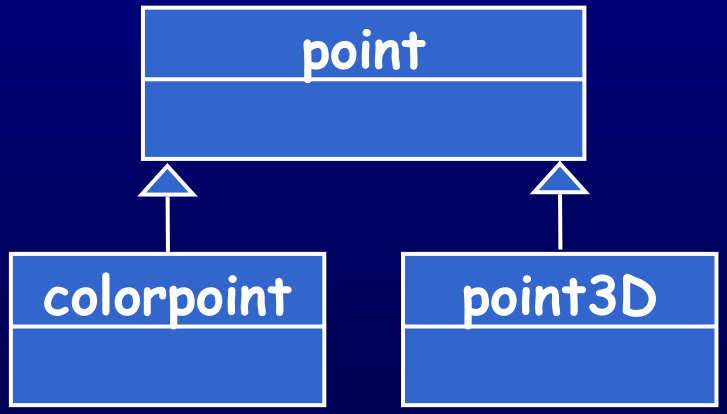

```
   point p = new point (0, 0);
   int x = (colorpoint) p.getX ();
Δ  int x = (point3D) p.getX ();
```

# 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***



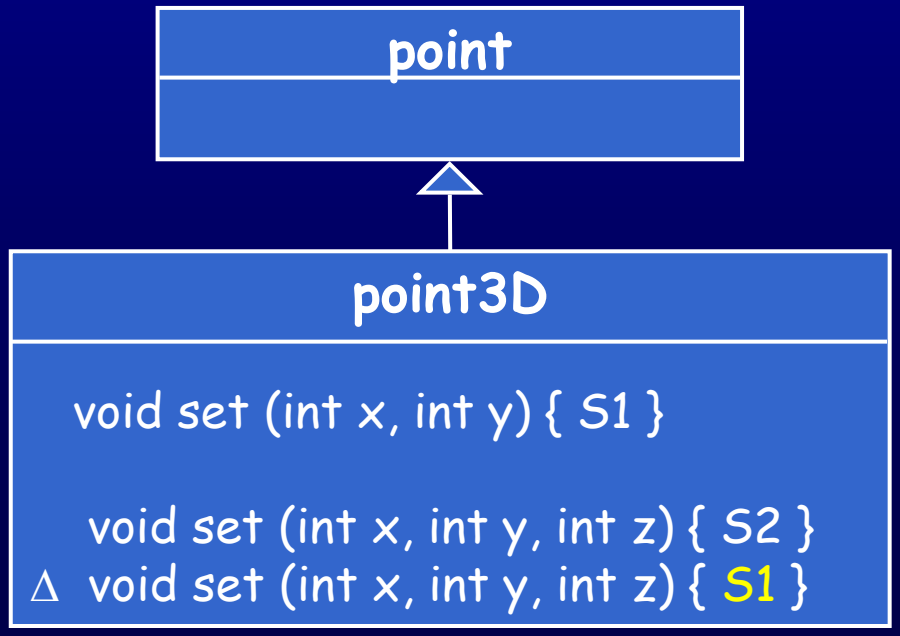

```
point p;
colorpoint cp = new colorpoint (0, 0);
point3D p3d = new point3D (0, 0, 0);
p = cp;
Δ p = p3d;
```

# OO Mutation Operators— *Polymorphism*

## 17. OMR — *Overloading Method Contents Replace*

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

***Example***

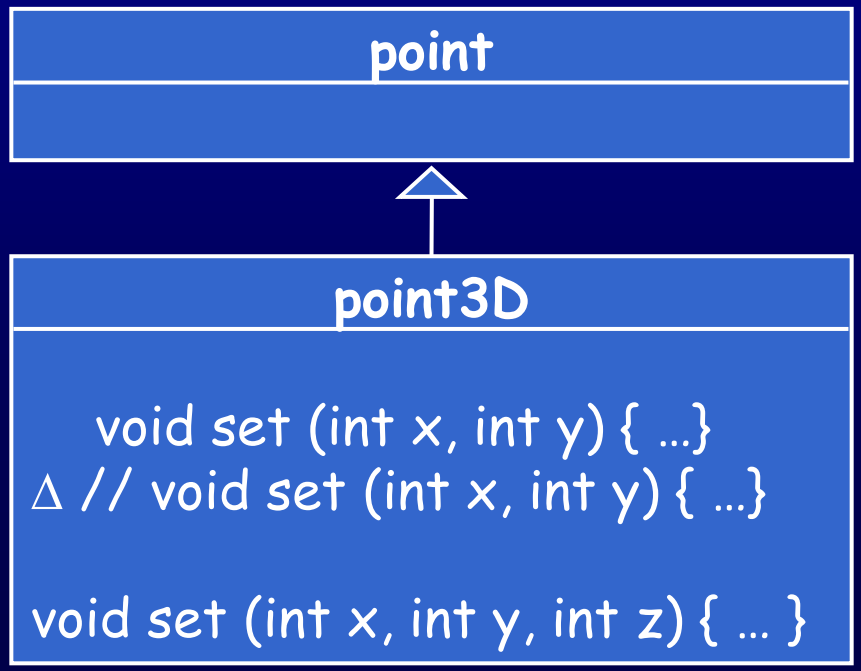


# OO Mutation Operators— *Polymorphism*

## 18. OMD — *Overloading Method Deletion*

Each overloaded method declaration is deleted, one at a time

***Example***

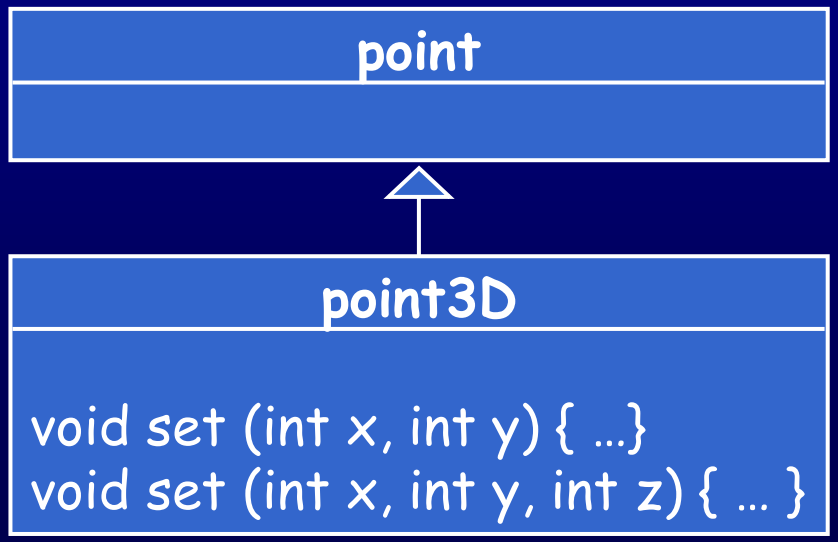


# OO Mutation Operators— *Polymorphism*

## 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);
```

# 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;
}
...
```

# 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;

    ...
```

# OO Mutation Operators—*Language Specific*

## 23. JSD — *Static Modifier Deletion*

Each instance of the `static` modifier is removed

***Example***

**point**

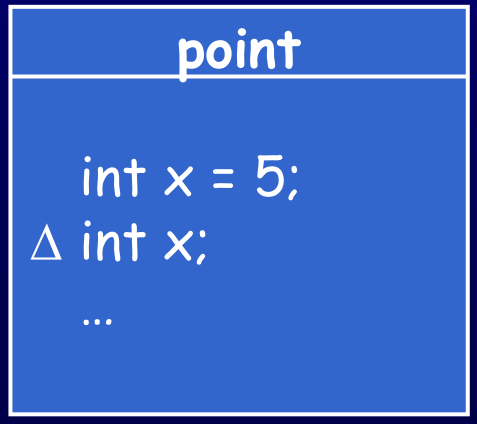
```
    public static int x = 0;
Δ1  public int x = 0;
    public static int Y = 0;
Δ2  public int y = 0;

    ...
```

# OO Mutation Operators—*Language Specific*

## 24. JID — *Member Variable Initialization Deletion*

Remove initialization of each member variable

***Example***

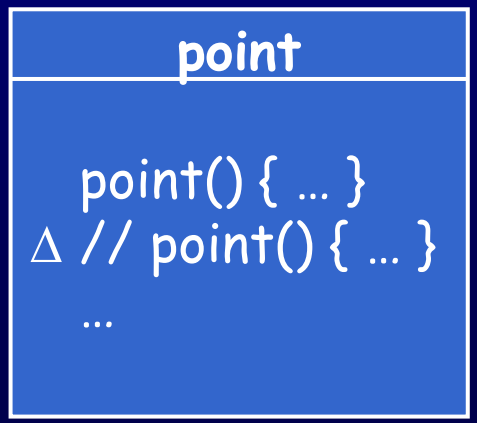


# OO Mutation Operators—*Language Specific*

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

Delete each declaration of default constructor (with no parameters)

***Example***



# Class Mutation Operators for Java

**(1) Encapsulation**

AMC

**(2) Inheritance**

HVD, HVI, IOD, OMM, OMR, SKD, PCD

**(3) Polymorphism**

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

**(4) Java-Specific**

TKD, SMC, VID, DCD

# 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/