Introduction to Software Testing
Chapter 5.3
Integration and Object-Oriented Testing

Paul Ammann & Jeff Offutt

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

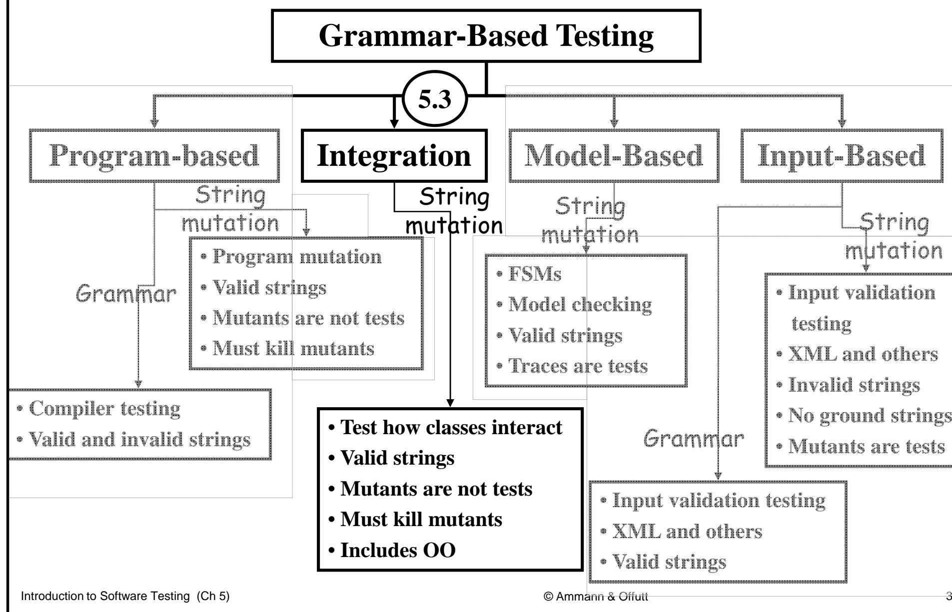
Integration and Object-Oriented Testing

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



Grammar Integration Testing (5.3.1)

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

Integration Mutation (5.3.2)

- **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 <u>calling</u> method by <u>modifying values that are sent</u> to a called method**

- **Change a <u>calling</u> method by <u>modifying the call</u>**

- **Change a <u>called</u> method by <u>modifying values that enter and leave</u> a method**
  - **Includes parameters as well as variables from higher scopes (class level, package, public, etc.)**

- **Change a <u>called</u> method by <u>modifying return statements</u> 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 objects.**

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**

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)**

Five Integration Mutation Operators (2)

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 ()"**

5. IREM — Integration Return Expression Modification

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

Integration Mutation Operators—Example

1. IPVR – Integration Parameter Variable Replacement

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

2. IUOI – Integration Unary Operator Insertion

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


Integration Mutation Operators—Example

3. IPEX – Integration Parameter Exchange

```
 Max (a, b);
Δ Max (b, a);
```

4. IMCD – Integration Method Call Deletion

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

5. IREM – Integration Return Expression Modification

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

Object-Oriented Mutation

Testing Levels
intra-method
inter-method
intra-class
inter-class

integration mutation operators

- **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

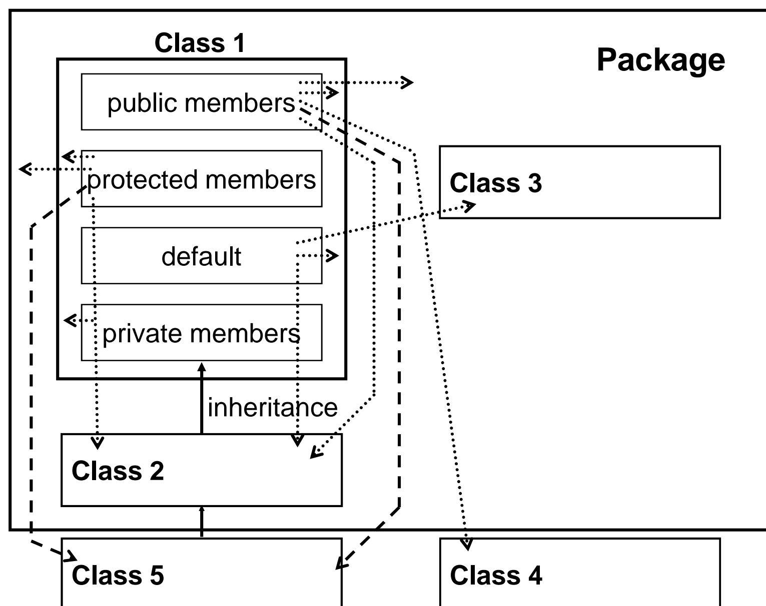
- **Encapsulation : An abstraction mechanism to implement information hiding, which is a design technique that attempts to protect parts of the design from parts of the implementation**
  - **Objects can restrict access to their member variables and methods**
- **Java provides four access levels (C++ & 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 <u>private</u>**
- **<u>Public</u> variables should seldom be used**
- **<u>Protected</u> variables are particularly <u>dangerous</u> – 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)



Object-Oriented 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**

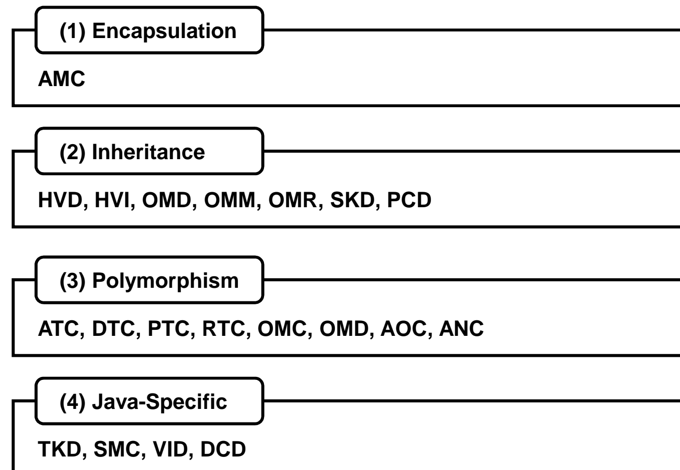
OO Language Feature Terms

- **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 <u>class</u> or <u>instance</u> variables and methods**
  - **<u>Static methods</u> can operate only on static variables; not instance variables**
  - **<u>Instance variables</u> 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



OO Mutation Operators—*Encapsulation*

1. AMC — Access Modifier Change

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

OO Mutation Operators—*Example*

1. AMC – Access Modifier Change

point

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

OO Mutation Operators—*Inheritance*

2. HVD — *Hiding Variable Deletion*

Each declaration of an overriding or hiding variable is deleted.

3. HVI — *Hiding Variable Insertion*

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

4. OMD — *Overriding Method Deletion*

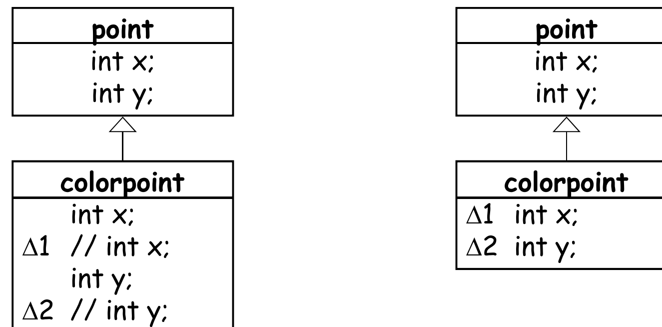
Each entire declaration of an overriding method is deleted.

5. OMM — *Overridden Method Moving*

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

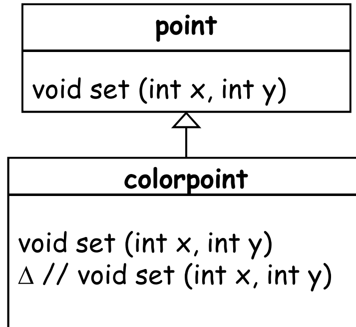
OO Mutation Operators—*Example*

2. HVD – Hiding Variable Deletion ***3. HVI – Hiding Variable Insertion***

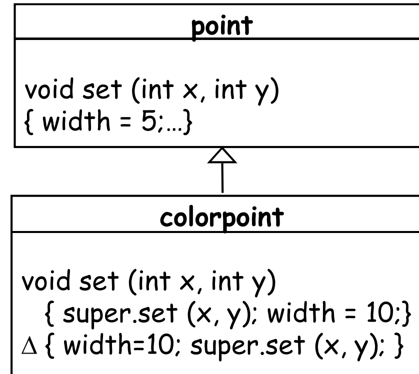


OO Mutation Operators—*Example*

4. OMD – Overriding Method Deletion



5. OMM – Overriding Method Moving



OO Mutation Operators—*Inheritance*

6. OMR — *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.

7. SKD — *Super Keyword Deletion*

Delete each occurrence of the `super` keyword.

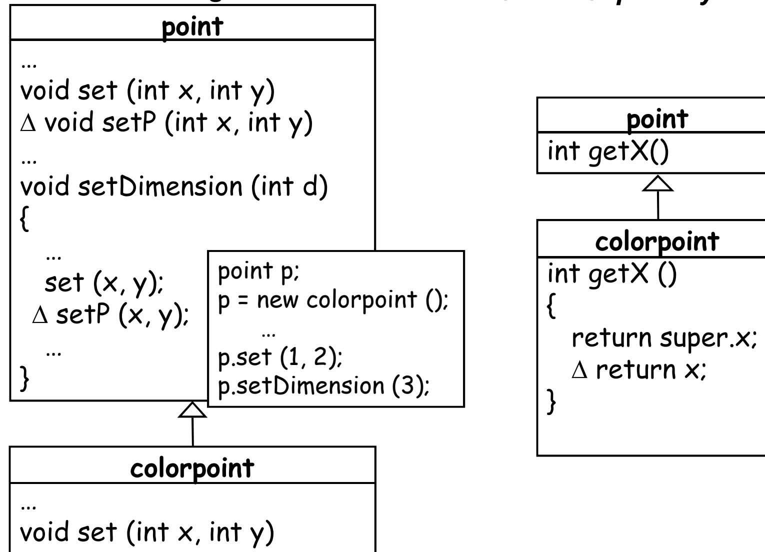
8. PCD —*Parent Constructor Deletion*

Each call to a `super` constructor is deleted.

OO Mutation Operators—*Example*

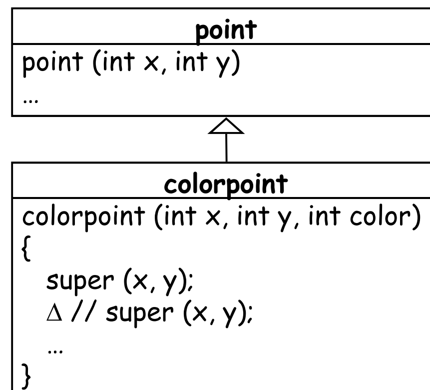
6. OMR – Overriding Method Rename

7. SKD – Super Keyword Deletion



OO Mutation Operators—*Example*

8. PCD – Parent Constructor Deletion



OO Mutation Operators—*Polymorphism*

9. ATC — *Actual Type Change*

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

10. DTC —*Declared Type Change*

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

11. PTC — *Parameter Type Change*

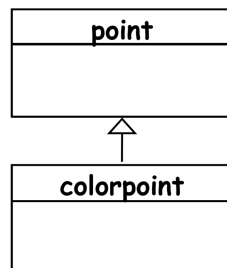
The declared type of each parameter object is changed in the declaration.

12. RTC — *Reference Type Change*

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

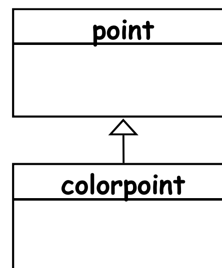
OO Mutation Operators—*Example*

9. ATC – Actual Type Change




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

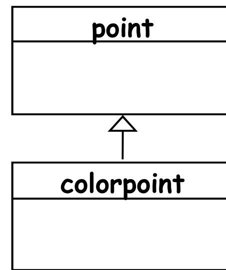
10. DTC – Declared Type Change




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

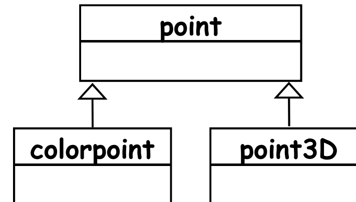
OO Mutation Operators—*Example*

11. PTC – Parameter Type Change




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

12. RTC – Reference Type Change




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


OO Mutation Operators—*Polymorphism*

13. OMC — *Overloading Method Change*

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

14. OMD — *Overloading Method Deletion*

Each overloaded method declaration is deleted, one at a time.

15. AOC — *Argument Order Change*

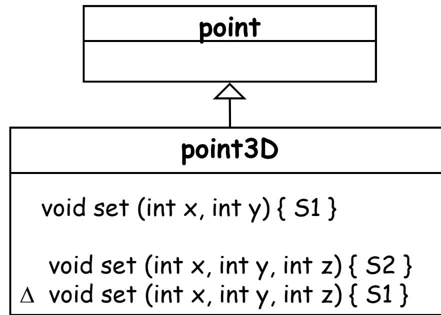
The order of the arguments in method invocations is changed to be the same as that of another overloading method, if one exists.

16. ANC — *Argument Number Change*

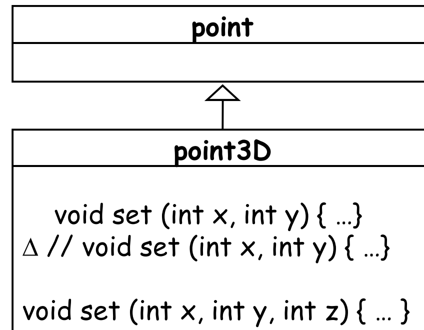
The number of the arguments in method invocations is changed to be the same as that of another overloading method, if one exists.

OO Mutation Operators—*Example*

13. OMR – Overloading Method Change

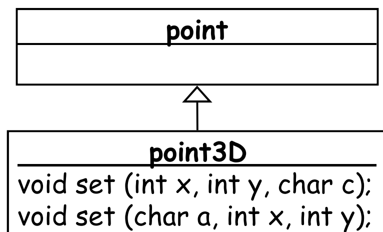


14. OMD – Overloading Method Deletion



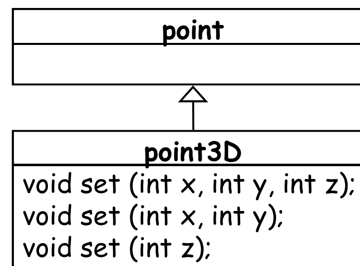
OO Mutation Operators—*Example*

15. AOC – Argument Order Change




```
  point3D p;
  p.set (1, 2, 't');
Δ p.set ('t', 1, 2);
```

16. ANC – Argument Number Change




```
  point3D p;
  p.set (1, 2, 3);
Δ p.set (2, 3);
Δ p.set (3);
```

OO Mutation Operators—*Language Specific*

17. TKD — *this Keyword Deletion*

Each occurrence of the keyword `this` is deleted.

18. SMC — *Static Modifier Change*

Each instance of the `static` modifier is removed, and the `static` modifier is added to instance variables.

19. VID —*Variable Initialization Deletion*

Remove initialization of each member variable.

20. DCD —*Default Constructor Delete*

Delete each declaration of default constructor (with no parameters).

OO Mutation Operators—*Example*

17. JTD – This Keyword Deletion

point

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

18. JSD – Static Modifier Change

point

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

   ...
```

OO Mutation Operators—*Example*

19. VID –Variable Initialization Deletion

```
point
   int x = 5;
Δ int x;
   …
```

20. DCD – Default Constructor Delete

```
point
   point() { … }
Δ // point() { … }
   …
```


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/**