Integration and Object Oriented Testing – Part 1

Dr. Isaac Griffith Idaho State University

Inspiration

“Lots of methodologies have come and gone, paradigms have changed but the requirements are always the same; Make it good and make it fast.” – Anonymous

Outcomes

After today’s lecture you will be able to:

- Understand the basic idea of integration testing and what it is for
- Understand the concepts of mutation testing applied to integration testing
- Understand and use the 4 basic types of mutation operators
- Understand and use the 5 basic integration mutation operators
- Start to understand the ideas of integration mutation applied to java and other OO languages

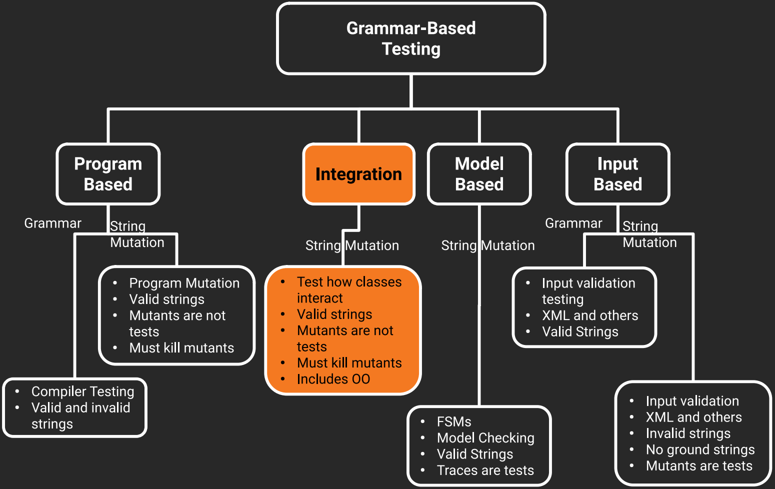
Integration and OO Testing

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



Grammar-Based Testing

Program Based

- Grammar
  - Compiler Testing
  - Valid and invalid strings
- String Mutation
  - Program Mutation
  - Valid strings
  - Mutants are not tests
  - Must kill mutants

Integration

- String Mutation
  - Test how classes interact
  - Valid strings
  - Mutants are not tests
  - Must kill mutants
  - Includes OO

Model Based

- String Mutation
  - FSMs
  - Model Checking
  - Valid Strings
  - Traces are tests

Input Based

- Grammar
  - Input validation testing
  - XML and others
  - Valid Strings
- String Mutation
  - Input validation
  - XML and others
  - Invalid strings
  - No ground strings
  - Mutants are tests

BNF Integration Testing

There is no known grammar testing at the integration level

Integration Mutation

- 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
  - Include parameters as well as variables from higher scopes (class level, package, public, etc.)
- Change a **called** method by **modifying return statements** from the method

5 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);
```

5 Integration Mutation Operators

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);
```

5 Integration Mutation Operators

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);
```

5 Integration Mutation Operators

4. IMCD – Integration Method Call Detection

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);
```

5 Integration Mutation Operators

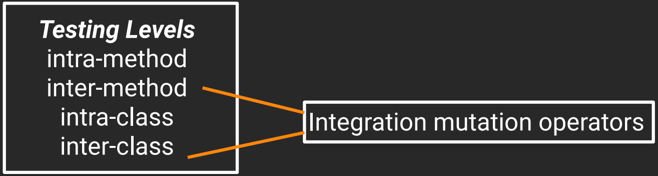
5. IREM – Integration Return Expression Modification

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



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 pars 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)

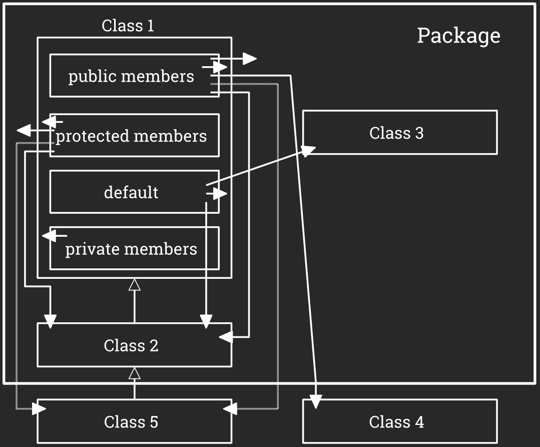
- Often **not used correctly** or understood, especially for programmers who are not well educated in **design**

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



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
- **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

Encapsulation Operators

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;
$\Delta 1$	public int x;
$\Delta 2$	protected int x;
$\Delta 3$	int x;

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

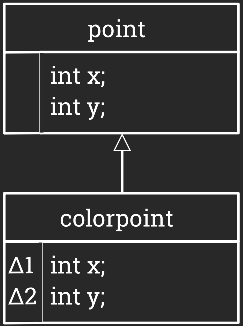
JTI, JTD, JSI, JSD, JID, JDC

Inheritance Operators

2. IHI – Hiding Variable Insertion

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

Example

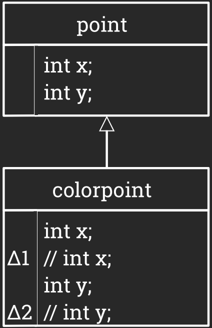


Inheritance Operators

3. IHD – Hiding Variable Deletion

Each declaration of an overriding or hiding variable is deleted

Example



point

```
int x;
int y;
```

colorpoint

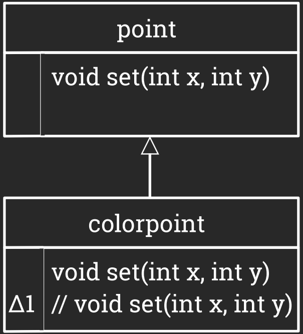
```
    int x;
Δ1  // int x;
    int y;
Δ2  // int y;
```

Inheritance Operators

4. IOD – Overriding Method Deletion

Each entire declaration of an overriding method is deleted

Example

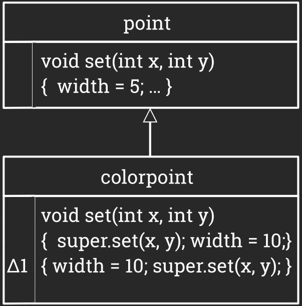


Inheritance Operators

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



point

void set(int x, int y)
{ width = 5; … }

colorpoint

void set(int x, int y)
{ super.set(x, y); width = 10;}
Δ1 { width = 10; super.set(x, y); }

For Next Time

- Review the Reading
- Review this Lecture
- Come to Class

Are there any questions?