REFLECTION

Instructors: Crista Lopes

Today’s Learning Objectives

- Reflection
  - Java
  - Python
- Class (code) loaders & reflection
  - How to load classes dynamically
  - How to instantiate and invoke objects reflectively
- Understand how software tools work
  - Live coding a unit test framework
- Learn about jar files

Reflection in Programming

- The ability to inspect, act, and even rewrite the code of the program **as the program runs.**

Reflection: Why?

- **Reflection** is used by programs which require the ability to examine or modify the runtime behavior.
  This is a relatively advanced feature and should be used only by developers who have a strong grasp of the fundamentals of programming.
  Reflection is a powerful technique and can enable applications to perform operations which would otherwise be impossible.

Practical Applications of Reflection

Proxies: e.g. a JDK Proxy of a large interface (20+ methods) to wrap (i.e. delegate to) a specific implementation. A couple of methods were overridden using an InvocationHandler, the rest of the methods were invoked via reflection.

Plugins: load specific classes at run-time.

Class Browsers and Visual Development Environments: A class browser has to be able to enumerate members of classes. Visual development environments can benefit from making use of type information available in reflection to aid code development.

Debuggers and Test Tools: Debuggers need to be able to examine private members on classes. Test harnesses can make use of reflection to systematically call a discoverable set APIs defined on a class, to insure a high level of code coverage in a test suite.

Reflection: who uses it

- Framework writers who want to be able to provide generic features for the framework's applications
  - Junit testing framework
  - Spring framework
- Software tool writers
  - IntelliJ, Eclipse, etc.
- Application programmers who want to automate boilerplate code

Outline

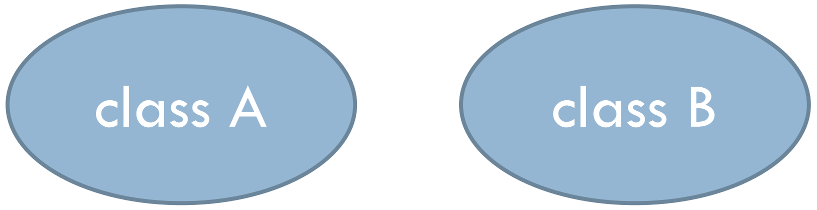
- **Reflection as used in**
  - **Java**
  - **Python**
- Some practical applications.
- Concerns to keep in mind while using Reflection.

Reflection in modern PLs

Program text:

```
class A {…}
class B {…}
Etc.
```

Runtime environment:



Etc.

Reflection in Java

- **java.lang.Class.**
  - **Class.forName()**

- **Class c = Class.forName(“edu.uci.inf212.Example");**

Reflection in Java

Object.getClass()

```
Class c = "foo".getClass();
```

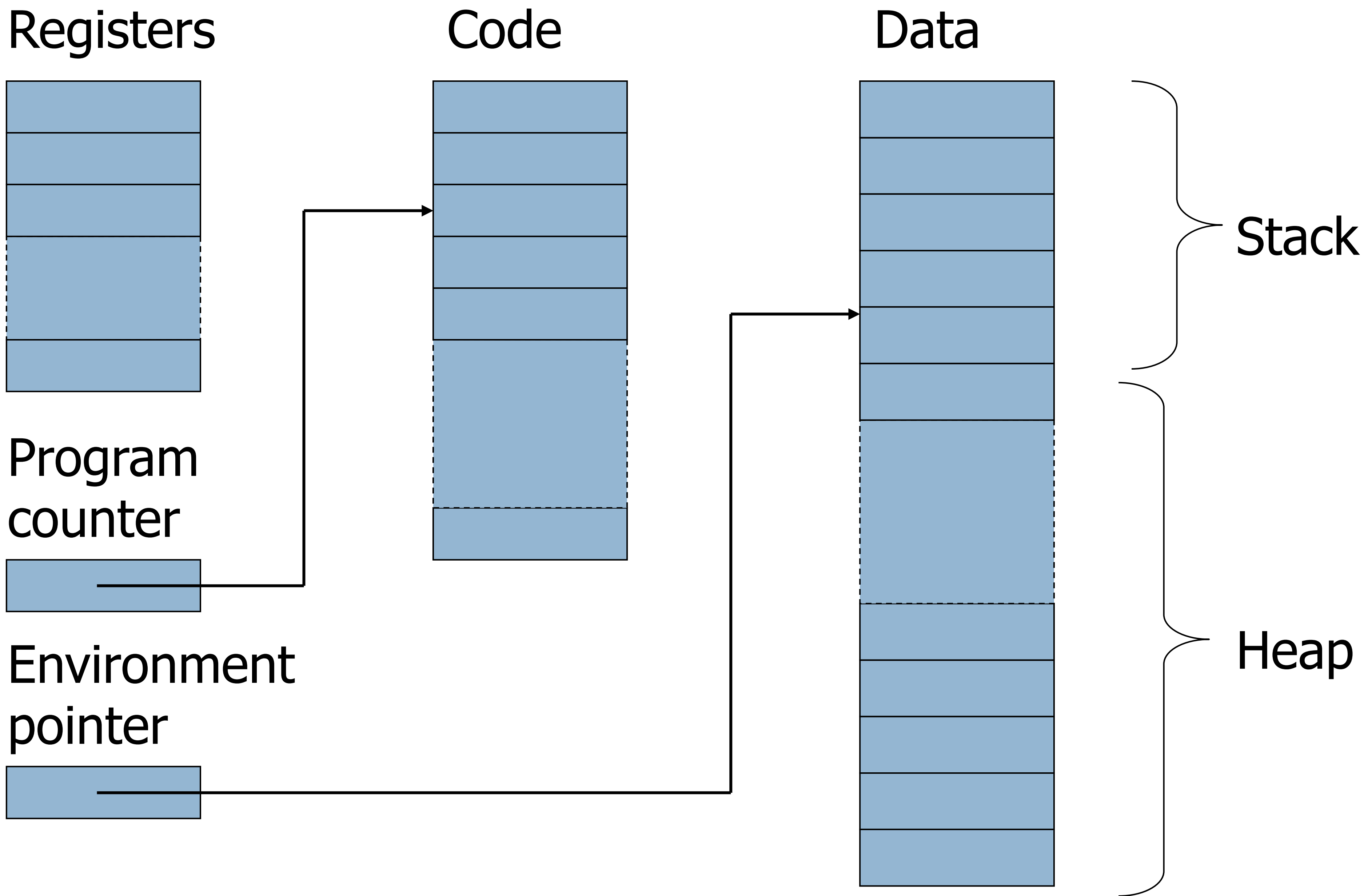
The .class Syntax()

```
Class c = boolean.class;
```

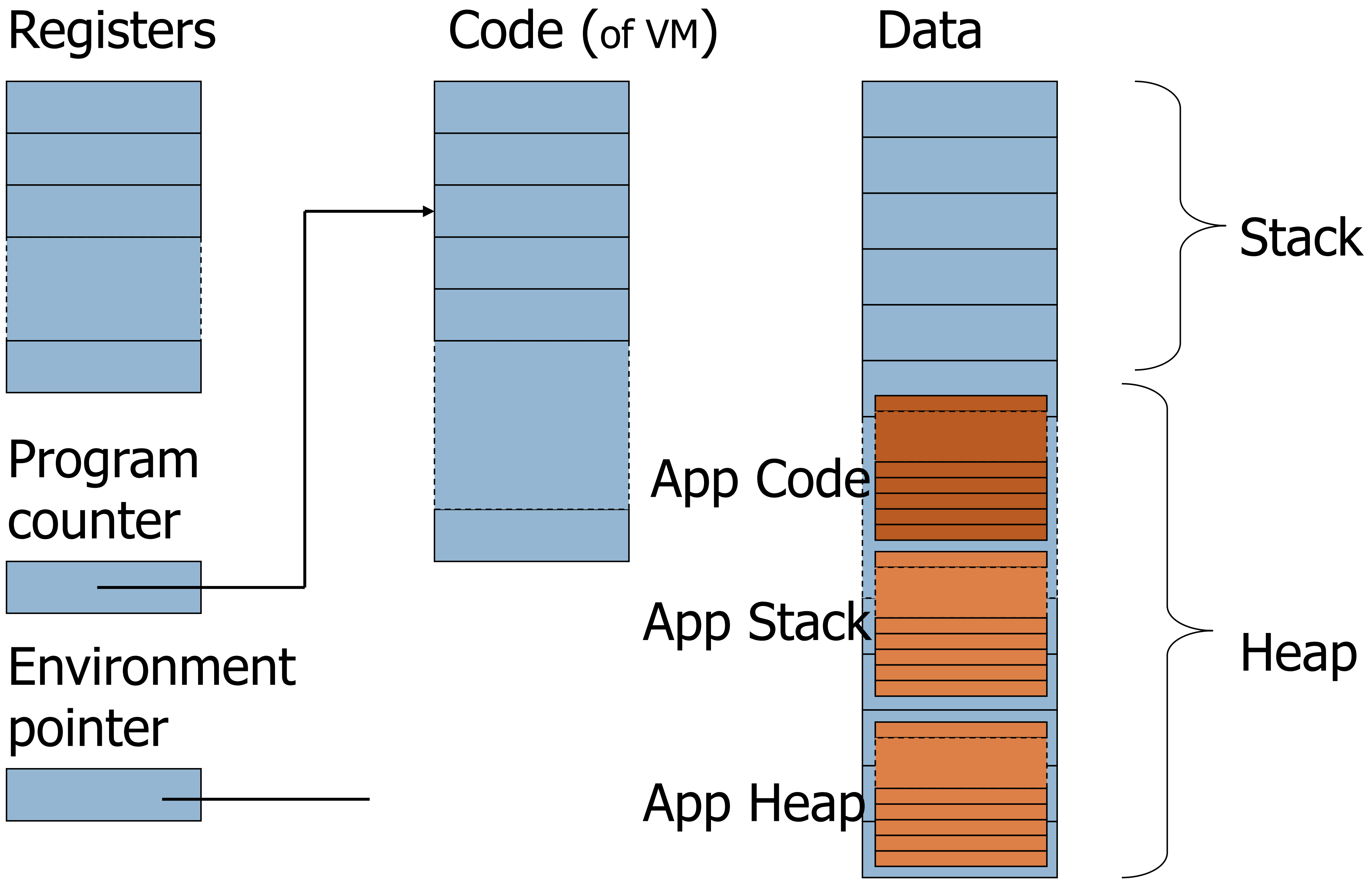
Live Coding

- Simple in-application reflection
- Reflecting on external classes: class loaders

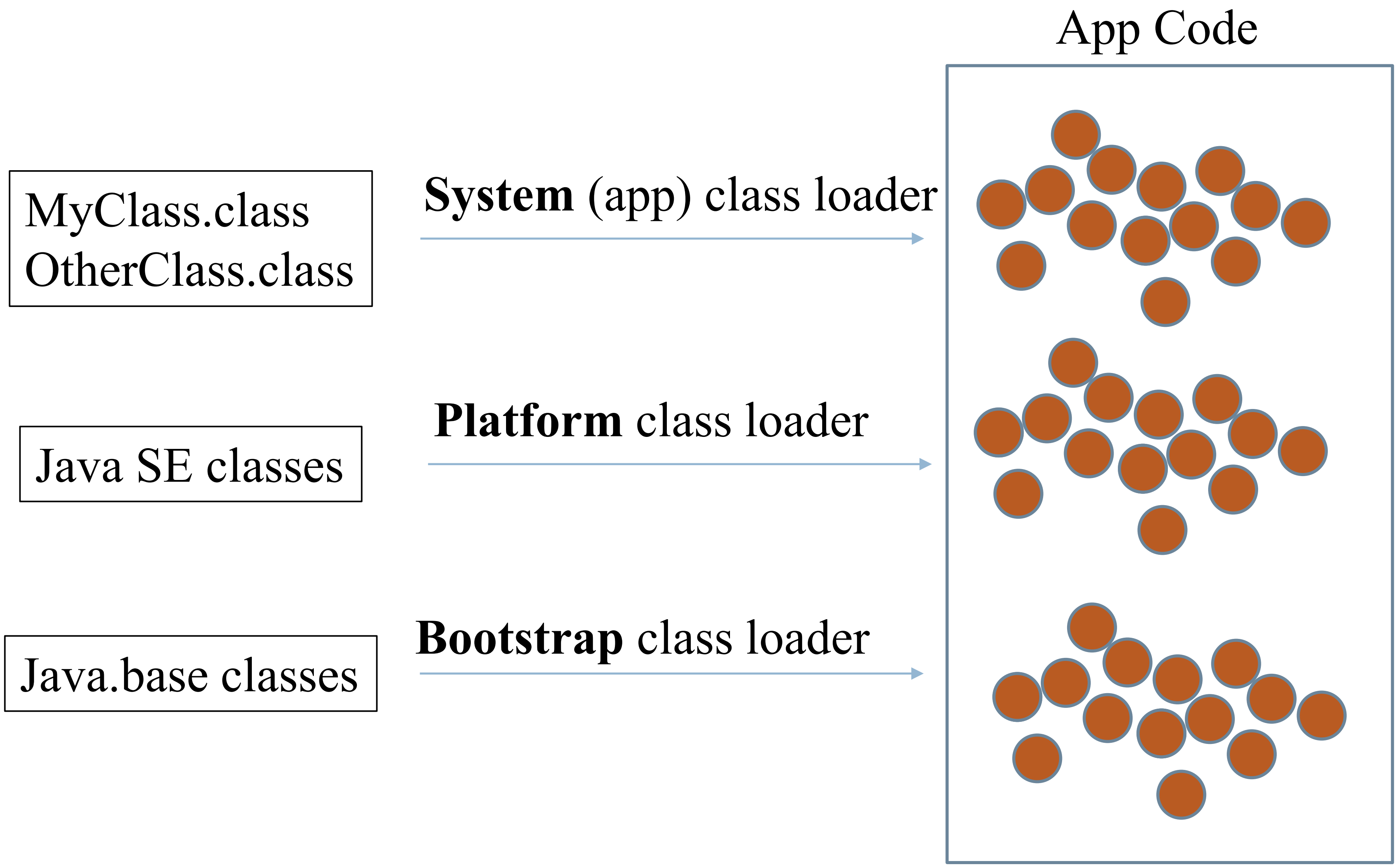
Simplified Machine Model



Virtual Machine



Class Loaders


App Code
MyClass.class
OtherClass.class
System (app) class loader
Java SE classes
Platform class loader
Java.base classes
Bootstrap class loader

Practical Use of Reflection

Frameworks

JUnit

- Unit tests for Java

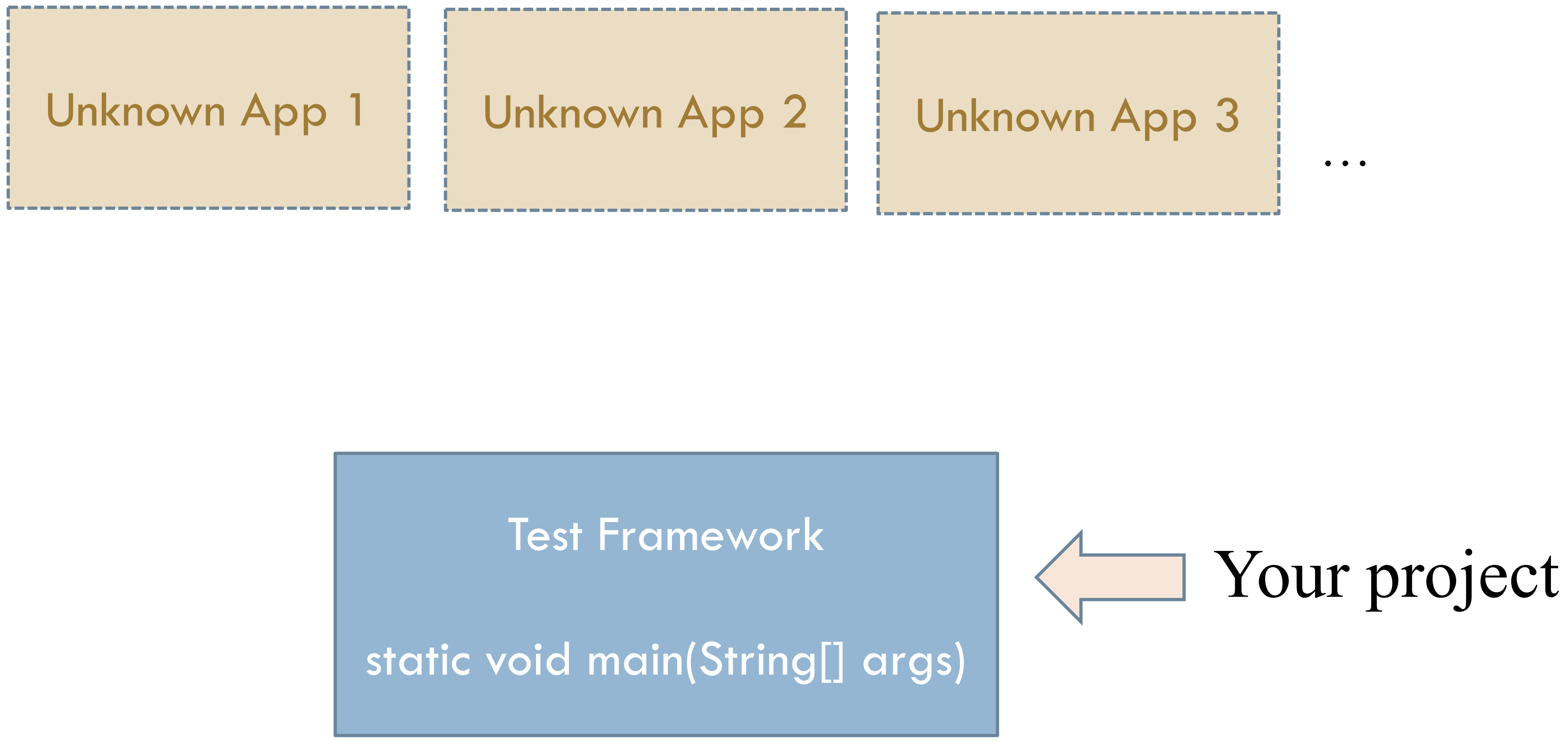
```
class CalculatorTest {
    Calculator calculator;

    @BeforeEach
    void setUp() {
        calculator = new Calculator();
    }

    @Test
    @DisplayName("Simple multiplication should work")
    void testMultiply() {
        assertEquals(20, calculator.multiply(4, 5),
                "Regular multiplication should work");
    }

    @RepeatedTest(5)
    @DisplayName("Ensure correct handling of zero")
    void testMultiplyWithZero() {
        assertEquals(0, calculator.multiply(0, 5), "should be zero");
        assertEquals(0, calculator.multiply(5, 0), "should be zero");
    }
}
```

How would you do it?



How do you call objects/methods that you don’t know anything about?!

Practical Use of Reflection

Software Development Tools

Practical Use of Reflection

Applications

Code and Databases: a difficult marriage

```
import java.sql.*;

class UpdateLogic {
  public static void main(String args[]) {
    Connection connection = null;
    try {
      Class.forName("imaginary.sql.iMsqlDriver");
      String url = "jdbc:msql://athens.imaginary.com:4333/db_test";
      Statement s;
      con = DriverManager.getConnection(url, "borg", "");
      con.setAutoCommit(false);     // make sure auto commit is off!
      s = con.createStatement();    // create the first statement
      s.executeUpdate("INSERT INTO t_test (test_id, test_val) " +
                      "VALUES(" + args[0] + ", '" + args[1] + "')");
      s.close();                    // close the first statement
      s = con.createStatement();    // create the second statement
      s.executeUpdate("INSERT into t_test_desc (test_id, test_desc) " +
                      "VALUES(" + args[0] +
                      ", `This describes the test.')");
      con.commit();                 // commit the two statements
      System.out.println("Insert succeeded.");
      s.close();                    // close the second statement
    }
    catch( SQLException e ) {
      if( con != null ) {...
```

Problems

- SQL statements as literals
  - No syntax checking, errors occur at runtime
- Object model (in memory) vs.
  Relational model (on disk)
  - Constantly having to parse/unparse data
- Duplication of knowledge of tables
  - Column names

Reflection to the rescue

```
public class MySQLGenericTableHandler<T>:MySqlFramework where T:class,new()
    public MySQLGenericTableHandler(string connectionString,
              string realm, string storeName) : base(connectionString){
        m_connectionString = connectionString;

        Type t = typeof(T);
        FieldInfo[] fields = t.GetFields(BindingFlags.Public |
                                         BindingFlags.Instance |
                                         BindingFlags.DeclaredOnly);
        if (fields.Length == 0)
            return;
        foreach (FieldInfo f in  fields)
        {
            if (f.Name != "Data")
                m_Fields[f.Name] = f;
            else
                m_DataField = f;
        }
    }
```

One Table

```
class MySqlGroupsGroupsHandler : MySQLGenericTableHandler<GroupData>
{
     public MySqlGroupsGroupsHandler(string connectionString,
                                     string realm,
                                     string store)
         : base(connectionString, realm, store)
     {
     }
}
```

```
public class GroupData
{
    public UUID GroupID;
    public Dictionary<string, string> Data;
}
```

Reflection to the rescue

In MySQLGenericTableHandler<T>

Generic Get

```
public virtual T[] Get(string[] fields, string[] keys)
{
    if (fields.Length != keys.Length)
        return new T[0];
    List<string> terms = new List<string>();

    using (MySqlCommand cmd = new MySqlCommand())
    {
        for (int i = 0 ; i < fields.Length ; i++)
        {
            cmd.Parameters.AddWithValue(fields[i], keys[i]);
            terms.Add("`" + fields[i] + "` = ?" + fields[i]);
        }
        string where = String.Join(" and ", terms.ToArray());
        string query = String.Format("select * from {0} where {1}",
                                     m_Realm, where);

        cmd.CommandText = query;
        return DoQuery(cmd);
    }
}
```

Reflection to the rescue

```
protected T[] DoQuery(MySqlCommand cmd) {
   List<T> result = new List<T>();
   using (MySqlConnection dbcon = new MySqlConnection(m_connectionString))
   {
       dbcon.Open();
       cmd.Connection = dbcon;

       using (IDataReader reader = cmd.ExecuteReader())
       {
           if (reader == null)
               return new T[0];

           CheckColumnNames(reader);

           while (reader.Read())
           {
               T row = new T();
```

Continued...

Reflection to the rescue

```
foreach (string name in m_Fields.Keys)
{
    if (reader[name] is DBNull)
        continue;
    if (m_Fields[name].FieldType == typeof(bool))
    {
        int v = Convert.ToInt32(reader[name]);
        m_Fields[name].SetValue(row, v != 0 ? true : false)
    }
    else if (m_Fields[name].FieldType == typeof(UUID))
        m_Fields[name].SetValue(row, DBGuid.FromDB(reader[r
    else if (m_Fields[name].FieldType == typeof(int))
    {
        int v = Convert.ToInt32(reader[name]);
        m_Fields[name].SetValue(row, v);
    }
    else
        m_Fields[name].SetValue(row, reader[name]);
}
```

Continued...

Reflection to the rescue

```
                if (m_DataField != null)
                {
                    Dictionary<string, string> data =
                        new Dictionary<string, string>();
                    foreach (string col in m_ColumnNames)
                    {
                        data[col] = reader[col].ToString();
                        if (data[col] == null)
                            data[col] = String.Empty;
                    }
                    m_DataField.SetValue(row, data);
                }
                result.Add(row);
            }
        }
    }
    return result.ToArray();
}
```

Performance Penalty

Reflection is powerful, but should not be used indiscriminately. If it is possible to perform an operation without using reflection, then it is preferable to avoid using it. The following concerns should be kept in mind when accessing code via reflection.

Performance Overhead : Because reflection involves types that are dynamically resolved, certain Java virtual machine optimizations can not be performed. Consequently, reflective operations have slower performance than their non-reflective counterparts, and should be avoided in sections of code which are called frequently in performance-sensitive applications.

Security

- **Security Restrictions :** Reflection requires a runtime permission which may not be present when running under a security manager. This is in an important consideration for code which has to run in a restricted security context, such as in an Applet.

- **Exposure of Internals :** Since reflection allows code to perform operations that would be illegal in non-reflective code, such as accessing private fields and methods, the use of reflection can result in unexpected side-effects, which may render code dysfunctional and may destroy portability. Reflective code breaks abstractions and therefore may change behavior with upgrades of the platform.