

Reflection

Computer Science and Engineering • College of Engineering • The Ohio State University

Lecture 28

Motivating Problem

Computer Science and Engineering • The Ohio State University

- Debugger/visualization tool
 - Takes an object, *any* object...
 - Displays the methods one can invoke on that object
- ```
int countMethods(Object o) {
 //return the number of methods o has...
 //how?
}
```
- Important library elements
    - Class java.lang.Class
    - Package java.lang.reflect

## The java.lang.Class Class

Computer Science and Engineering • The Ohio State University

- Every class extends java.lang.Object
- Every object is an instance of a class
- For every class, there is an instance of java.lang.Class
  - You can get an *object* that contains everything you might need to know about a class
  - Of course, Class inherits from Object
- This class is generic, ie Class<T>
  - T is the type represented
    - eg Class<String> for String
  - Use Class<?> if type is unknown (typical)

## Obtaining a Class Instance: 1

Computer Science and Engineering • The Ohio State University

- If an instance of class exists, use getClass()
  - A method in java.lang.Object

```
Class<? extends X> getClass();
```
  - Where X is (erasure of) *declared* type of instance

```
int countMethods (Number n) {
 Class<? extends Number> c = n.getClass();
}
```
- Example

```
Class<?> c = "foo".getClass();
```
- Example

```
Employee e = new Employee();
Class<?> c = e.getClass();
```

## Obtaining a Class Instance: 2

Computer Science and Engineering • The Ohio State University

- ❑ Special syntax: Class name followed by `.class`
  - `Class<?> c = Employee.class;`
  - Looks like a static field of `java.lang.Object`
  - But actually a special syntactic construct
- ❑ Does not require an instance
  - But does require the class name to be known at *compile* time
- ❑ Works for primitive types too
  - `Class<?> c = boolean.class;`
  - The `getClass()` approach does *not* work for primitives

## Obtaining a Class Instance: 3

Computer Science and Engineering • The Ohio State University

- ❑ Class's static method `forName()` returns a `Class` based on a string
  - `static Class<?> forName(String className);`
  - Example
    - `Class<?> c = Class.forName("Employee");`
- ❑ Fully dynamic:
  - Class name (ie `Employee`) not known at compile time
  - No instances of class (`Employee`) exist
- ❑ Throws `ClassNotFoundException` if a class with the given name can not be located

## Using an Instance of java.lang.Class

Computer Science and Engineering • The Ohio State University

- To get the class name (a String)  
`String getName();`
- To get modifiers (public, abstract, final)
  - eg public, final, abstract, interface
  - Encoded as an integer  
`int getModifiers();`
  - Use utility class Modifiers to decode the int
- To get superclass
  - Returns direct parent  
`Class<? super T> getSuperclass();`
- To get interfaces
  - Returns array of implemented interfaces  
`Class<?>[] getInterfaces();`

## Example

Computer Science and Engineering • The Ohio State University

```
public class GetClassExample {
 public static void main(String[] args) {
 Object e = new Employee();
 Class<?> c = e.getClass()

 System.out.println("Class name: " + c.getName());
 System.out.println("Class super class: " + c.getSuperclass());

 int m = c.getModifiers();
 System.out.println("Class is public: " + Modifier.isPublic(m));
 System.out.println("Class is final: " + Modifier.isFinal(m));
 System.out.println("Class is abstract: "+Modifier.isAbstract(m));
 }
}

$ java GetClassExample
Class name: Employee
Class super class: class java.lang.Object
Class is public: true
Class is final: false
Class is abstract: false
```

## Calling the 0-Argument Constructor

Computer Science and Engineering • The Ohio State University

- Use method `newInstance()` on a `Class<T>` instance

```
T newInstance();
```

  - Often use `Object` for return type

```
Class<?> c = Class.forName("Simple");
Object x = c.newInstance();
System.out.println(x.toString());
```
- For this to work, class (eg `Simple`) must have a public 0-argument constructor
  - Otherwise, an `InstantiationException` is thrown

## Example

Computer Science and Engineering • The Ohio State University

```
public class NewInstanceExample {
 public static void main(String[] args) throws
 ClassNotFoundException, InstantiationException,
 IllegalAccessException {
 Class<?> c = Class.forName(args[0]);
 Object o = c.newInstance();
 System.out.println("Just made: " + o);
 }
}
```

```
$ java NewInstanceExample Employee
Just made: Employee: John Smith 50000
```

```
$ java NewInstanceExample java.util.Date
Just made: Mon Dec 1 9:25:20 EST 2008
```

## Calling Constructor with Arguments

Computer Science and Engineering • The Ohio State University

- Two steps:
  - Get an object representing the constructor
    - See `java.lang.reflect.Constructor<T>`
    - Use `getConstructor` method in `Class`  
`Constructor<T>`  
`getConstructor(Class<?>... pTypes);`
  - Call `newInstance` on that constructor  
`T newInstance(Object... args);`
- Varargs syntax (...) allows for an arbitrary number of arguments of that type
  - Syntactic sugar for an array of that type

## Example

Computer Science and Engineering • The Ohio State University

```
Class<?> c = Class.forName("Employee");

Class<?>[] paramTypes = {
 String.class,
 String.class,
 int.class };

Constructor<?> cons = c.getConstructor(paramTypes);
System.out.println("Found the constructor: " + cons);

Object[] args = {
 "Fred",
 "Flintstone",
 new Integer(9000) };

Object o = cons.newInstance(args);
System.out.println("Just made: " + o);
```

## Discovering a Class's Members

Computer Science and Engineering • The Ohio State University

- Members are represented by instances of
  - `java.lang.reflect.Field`
  - `java.lang.reflect.Method`
  - (and `java.lang.reflect.Constructor<T>`)
- Two kinds of methods
  - Those that enumerate members
  - Those that return a specific member (eg based on parameter types)
- Some include inherited members
- Some include private members

## Example

Computer Science and Engineering • The Ohio State University

```
Class<?> c = Class.forName("Employee");
Method[] methods = c.getMethods();

for(Method m : methods) {
 System.out.println("Found: " + m);
}

Found: public java.lang.String Employee.toString()
Found: public int Employee.getSalary()
Found: public void Employee.setSalary(int)
Found: public native int java.lang.Object.hashCode()
Found: public final native java.lang.Class
 java.lang.Object.getClass()
Found: public boolean
 java.lang.Object.equals(java.lang.Object)
Found: public java.lang.String
 java.lang.Object.toString()
... etc (other Object methods)
```

# Taxonomy for Member Discovery

Computer Science and Engineering ■ The Ohio State University

| Class API *                       | List? | Inherited? | Private? |
|-----------------------------------|-------|------------|----------|
| <code>getDeclaredMethod()</code>  | no    | no         | yes      |
| <code>getMethod()</code>          | no    | yes        | no       |
| <code>getDeclaredMethods()</code> | yes   | no         | yes      |
| <code>getMethods()</code>         | yes   | yes        | no       |

\*similar for Field and Constructor

## Example

Computer Science and Engineering ■ The Ohio State University

```
Class<?> c = Class.forName("Employee");
Class<?>[] pTypes = {int.class};
Method m = c.getMethod("setSalary", pTypes);
System.out.println("Found: " + m);
```

## Accessing Members

Computer Science and Engineering • The Ohio State University

- ❑ Call a method using invoke

```
Object invoke(Object obj, Object... args);
```

  - First parameter is object instance on which method is being invoked (ie the target)
  - Subsequent varargs are parameters for the method invocation
  - Returned object is return value resulting from method invocation
- ❑ Ordinary polymorphism
  - Dynamic type of obj determines code that is called
- ❑ Exceptions?
  - Can throw InvocationTargetException if underlying method throws an exception
  - Underlying exception is chained to this one as its cause

## Example

Computer Science and Engineering • The Ohio State University

```
Class<?> c = Class.forName("Employee");
Class<?>[] pTypes = {int.class};
Method m = c.getMethod("setSalary", pTypes);
Object theObject = c.newInstance();
Object[] parameters = {new Integer(90000)};
m.invoke(theObject, parameters);
```

- ❑ Note:
  - Above code works (without recompilation) for *any* class with:
    1. A 0-argument constructor (for newInstance call)
    2. A single-parameter (int) method
  - Just use different strings (eg "Student", and "addCredits")
  - Does not rely on inheritance relationship or polymorphism

# Security

Computer Science and Engineering • The Ohio State University

- Reflection allows you to see private members
  - `getDeclaredMethods()` returns *all* methods, even private ones!
- All members (Field, Method, Constructor) extend `java.lang.reflect.AccessibleObject`
  - Can query whether member is visible  
`boolean isAccessible();`
  - Can also set/change visibility!  
`void setAccessible(boolean flag);`
- Thus, reflection allows you full access to private members
  - Invoke private methods
  - Read and write private fields

# Summary

Computer Science and Engineering • The Ohio State University

- Class objects
  - Correspond to types in the program
  - Obtained from instances, class names, or strings
  - Provide basic information about class
  - Provide members of class
- Instantiation
  - Through Class's `newInstance()` – no arguments
  - Through Constructor's `newInstance()`
- Members
  - Lists or individual members
  - Methods can then be invoked
  - Fields can be accessed
  - Privacy can be undermined