Reflection

Background

- Turing's great insight: programs are just another kind of data
  - Source code is text
  - Manipulate it line by line, or by parsing expressions
- Compiled programs are data, too
  - Integers and strings are bytes in memory that you interpret a certain way
  - Instructions in methods are just bytes too
- No reason why a program can't inspect itself, i.e., read its own data

How Objects Work

class Point {
    public Point(int x, int y)
    {
    }
    public getX() {...}
    public getY() {...}
    protected int x, y;
}

The Class class

- Instances of the class Class store information about classes
  - Class name
  - Inheritance
  - Interfaces implemented
  - Methods, members, etc.
- Can look up instances:
  - By name
  - From an object
Showing a Type

```java
public static void showType(PrintStream out, String className) throws ClassNotFoundException {
    Class thisClass = Class.forName(className);
    String flavor = thisClass.isInterface() ? "interface" : "class";
    out.println(flavor + " " + className);
    Class parentClass = thisClass.getSuperclass();
    if (parentClass != null) {
        out.println("extends " + parentClass.getName());
    }
    Class[] interfaces = thisClass.getInterfaces();
    for (int i=0; i<interfaces.length; ++i) {
        out.println("implements " + interfaces[i].getName());
    }
}
```

Output for Type Example

```java
class java.lang.Object
extends java.util.AbstractMap
implements java.util.Map
implements java.lang.Cloneable
implements java.io.Serializable

class Point
extends java.lang.Object
```

Examining Class Contents

```java
public static void showContents(PrintStream out, boolean hideObject, String name) throws ClassNotFoundException {
    Class cls = Class.forName(name);
    out.println(name);
    showMembers(out, hideObject, name + " fields", cls.getFields());
    showMembers(out, hideObject, name + " constructors", cls.getConstructors());
    showMembers(out, hideObject, name + " methods", cls.getMethods());
}
```

Examining Class Contents

```java
public static void showMembers(PrintStream out, boolean hideObject, String title, Member[] members) {
    out.println(" " + title);
    for (int i=0; i<members.length; ++i) {
        if (members[i].getDeclaringClass() == Object.class && hideObject) {
            continue;
        }
        out.println("\t" + members[i]);
    }
}
```
Point
Point fields
Point constructors
public
Point(java.lang.String, int, int)
Point methods
public java.lang.String
Point.toString()
public java.lang.String
Point.getName()
public void
Point.setName(java.lang.String)
public int
Point.getX()
public void Point.setX(int)
public int Point.getY()
public void Point.setY(int)

Getting at Members

• How to access members of a specific object?
  – Without making raw pointers into memory part of
    the language
  – They are a rich source of errors in C/C++
• Introduce a class Field
  – Encapsulates access to a particular field of
    instances of a class
  – Knows "where the field is" in objects of that class
  – Use its get() and set() methods to inspect and
    modify the object

Examining Fields

public static void main(String[] args) {
    PublicPoint p = new
    PublicPoint("center", 3, 3);
    showField(System.out, p, "fName");
    showField(System.out, p, "fX");
    showField(System.out, p, "fY");
    showField(System.out, p, "fZ");
    
    public static void showField(PrintStream out,
        Object obj, String fieldName) {
        try {
            Class cls = obj.getClass();
            Field field = cls.getField(fieldName);
            Object value = field.get(obj);
            out.println(fieldName + ": " + value);
        } catch (NoSuchFieldException e) {
            System.err.println(e);
        } catch (IllegalAccessException e) {
            System.err.println(e);
        }
    }
}
public static void showMethods(
    PrintStream out, Object obj
) throws NoSuchMethodException,
    InvocationTargetException, 
    InvocationTargetException {
    Class cls = obj.getClass();
    out.println(cls.getName());
    Member[] members = cls.getMethods();
    for (int im=0; im<members.length; ++im) {
        Method meth = (Method)members[im];
        if (meth.getDeclaringClass() == cls) {
            showMethod(out, (Method)members[im]);
        }
    }
}

Output

fName: center
fX: 3
fY: 3
java.lang.NoSuchFieldException: fZ

Calling Methods

1) Look up a method based on its name and signature
2) Call a method, passing in parameters and capturing return value
   - Specify a signature as an array of Class objects
     - Specifies the types of arguments
     - Special values for types like int and boolean
     - Note: cannot select based on return type
   - Specify parameters as an Object array
     - Use Integer instead of int, etc.
     - Java will extract values as necessary

Switching on Type

- Often have to handle basic types case-by-case.
- Pattern:
  - Inspect Object to find out what type it is
  - Cast it to that type
  - Do something with integers
  - Something else with strings
  - Everything else expressed in terms of these
Reflection in Python

• Special attributes in a Python object:
  >>> c = C(20)
  >>> c.__class__
  <class '__main__.C' at 0x53870>
  >>> c.__dict__
  {'x': 20}
  >>> c.__module__
  '__main__'
  >>> C.__name__
  'C'
  >>> c.__class__.__name__
  'C'

Built-in methods

• getattr(object, name)
  – returns the value of the attribute name
• hasattr(object, name)
  – returns true if the object has an attribute by the given name
• setattr(object, name, value)
  – assign value to attribute name
• type(object)
  – returns the type of the object

Invoking a method given an object

class C:
    def __init__(self, val=-1):
        self.x = val
    def foo(self):
        print self.x
if __name__ == '__main__':
    c = C(10)
f = getattr(c, "foo")
f()
f = getattr(c, "x")
print f

Loading a class

(see the example posted on the web page)
Key Points

• There is no magic
  – A class is just a data structure
  – A method is just a data structure, too
• It just happens to contain bytes that look like instructions for the interpreter
• The call stack is another data structure
  – With libraries to give you access to it at runtime
• Many programming tools make use of reflection
  – We'll see one in the next lecture