Reflection

a.k.a. Introspection

Reflection

- Allows runtime discovery of an object's
  - fields
  - constructors
  - methods
- Particularly useful for creating tools such as debuggers and class browsers
- Within access restrictions can

  public, protected, "package", private
  - get and set discovered fields
  - invoke discovered methods
  - create new objects using discovered constructors
- Supported by
  - new methods in the class Class
  - new classes
    - Member, Field, Constructor, Method, Array, Modifier

Reflection Example

```
// Read an object from a persistent store.
// This is one example of where reflection is useful.
FileInputStream fis = new FileInputStream(“save.jos”);
ObjectInputStream ois = new ObjectInputStream(fis);
Object object = ois.readObject();

Class clazz = object.getClass();

// Get public information.
Field[] fields = clazz.getFields();
Constructor[] constructors = clazz.getConstructors();
Method[] methods = clazz.getMethods();

// Get all information.  Only “trusted” code can do this.
// SecurityManager.checkMemberAccess can throw
// SecurityException to prevent access to this information.
Field[] fields = clazz.getDeclaredFields();
Constructor[] constructors = clazz.getDeclaredConstructors();
Method[] methods = clazz.getDeclaredMethods();

System.out.println(“Object type is “ + class.getName());
System.out.println(“Field types and names are”);
for (int i = 0; i < fields.length; ++i) {
    Field field = fields[i];
    System.out.println(field.getType().getName() + " " +
                       field.getName());
}
```

Highlights of Reflection Methods in java.lang.Class

- Discovery of class attributes
  - `String getName()`
  - `Class getSuperclass()`
  - `Class[] getInterfaces()`
  - `int getModifiers()`
    - returns a mask of modifier bits representing abstract, final, interface, native, private, protected, public, static, synchronized, transient, volatile
    - also used for fields, constructors, and methods

not part of the persistent state of an object (sometimes can be calculated from non-transient fields)

The field is used by multiple synchronized threads so the compiler should read its value from memory for every access instead of optimizing access by saving a copy on the stack.

- Discovery of fields
  - `Field[] getFields()` - only public
  - `Field[] getDeclaredFields()` - all
- Discovery of constructors
  - `Constructor[] getConstructors()` - only public
  - `Constructor[] getDeclaredConstructors()` - all
- Discovery of methods
  - `Method[] getMethods()` - only public
  - `Method[] getDeclaredMethods()` - all

Highlights of Reflection Methods in java.lang.Class (Cont'd)

- Creating a new object
  - `Object newInstance()`
  - uses no-arg constructor
  - throws IllegalAccessException if caller has insufficient access to use the no-arg constructor
- Creating a class object
  - using `forName()`
    - `Class clazz = Class.forName("java.util.Vector");`
    - throws ClassNotFoundException
  - using `.class`
    - `Class clazz = java.util.Vector.class;`

Highlights of Interface java.lang.reflect.Member

- Implemented by the classes Field, Constructor, and Method which are all types of members
- `Class getDeclaringClass()`
- `String getName()`
- `int getModifiers()`
  - returns a mask of modifier bits representing abstract, final, interface, native, private, protected, public, static, synchronized, transient, volatile

Highlights of Class java.lang.reflect.Field

in Member

- `Class getDeclaringClass()`
- `String getName()`
- `int getModifiers()`

- `Class getType()`
  - tells the type of the field
- `Object get(Object object)`
  - gets the value of the field for a given object
  - returns primitive values in type wrapper objects
- `void set(Object object, Object value)`
  - sets the value of the field for a given object
  - put primitive values in type wrapper objects

object containing the field

Highlights of Class java.lang.reflect. Constructor

in Member

- `Class getDeclaringClass()`
- `String getName()`
  - same for all constructors in the same class
- `int getModifiers()`

- `Class[] getParameterTypes()`
  - tells the parameters types the constructor requires
- `Class[] getExceptionTypes()`
  - tells which exceptions the constructor may throw
- `Object newInstance(Object[] args)`
  - uses the constructor to create a new instance using the given arguments

Highlights of Class java.lang.reflect.Method

in Member

- `Class getDeclaringClass()`
- `String getName()`
- `int getModifiers()`

- `Class getReturnType()`
  - tells the type of the method return value
- `Class[] getParameterTypes()`
  - tells the parameters types the method requires
- `Class[] getExceptionTypes()`
  - tells which exceptions the method may throw
- `Object invoke`
  `(Object obj, Object[] args)`

object supporting the method

method return value

  - invokes the method on a given object passing `args` to it
    - use type wrapper objects for primitive arguments
  - throws InvocationTargetException if obj doesn't support the method
  - passing Method objects provide the same capability as passing function pointers in C/C++
    - also consider using anonymous inner classes

Lab

- Complete ClassBrowser
  - copy ClassBrowserStub.java, AlphabeticalList.java, Constrainable.java and GridBagPanel.java in \\Duke\JavaAdv\Reflection\ClassBrowser to your directory
    - only ClassBrowserStub requires changes
  - look for comments indicating missing code (//>) and insert calls to reflection methods
  - compile with `javac -depend ClassBrowser.java`
  - why is recursion necessary in getTypeString()?