Assignment 3: Advanced Squeak

Introduction

1. The assignment is due on 31.5.15 at 23:55.
2. Objective of this assignment: Cultivating a deeper understanding of Squeak and the 5 level object model using Squeak capabilities such as class creation and interception of messages sent to Squeak objects. Using these capabilities you will implement multiple inheritance using inheritance specifiers.
3. The TA in charge of this assignment is Nurit Moscovici. Questions should be sent to the mail nuritm@cs.technion.ac.il with the subject “236703 HW3”.
4. All classes should belong to a category called “OOP3”.
5. In order to avoid mistakes please read the FAQ section in the course website which will be updated on an ongoing basis.
6. This excersise must be implemented in the Squeak version 4.5!

Inheritance Specifiers

C++ enables the user to add specifiers (public/private/protected) to class inheritance. We want to add a similar property to Squeak. The subject of inheritance specifiers will be studied in depth later on in this semester, but we will give a short summary for the purposes of the assignment. In this assignment we consider all methods in Squeak to be “public”.

Adding specifiers to inheritance influences which classes can know that the child class inherits from the parent class. For example, given a class B that inherits from class A with the specifier:

- **Private:** Only B itself “knows” that B inherits from A. That is to say only instances of class B itself can execute methods defined in the sub-object A through an instance of B. All other classes do not see the inheritance. Another way of looking at it is that all methods that are public/protected in A are considered private in B.

- **Protected:** Only B and classes that inherit from B “know” that B inherits from A, so only an instance of one of these classes can execute methods defined in the sub-object A through an instance of B. All methods that are public/protected in A are considered protected in B and classes inheriting from B.

- **Public:** All of the classes “know” that B inherits from A, and so all classes can execute methods defined in the sub-object A through an instance of B.

You can see examples specific to this assignment in “hw3 examples.pdf”. You can read more about the subject here (but please note that this assignment does not require you to implement all of the details of the C++ mechanism. The mechanism defined in this assignment is simpler, and is described more in depth in the examples pdf and in the rest of the assignment.
Part A – Helper Classes
The class Parent will be used in the multiple inheritance data structure defined in parts B and C.

Instance Variables:

a) parentClass - Holds the class we wish to inherit from. Initialized to Object.
b) inheritanceType - Holds a string defining the inheritance specifier. The possible values of the string are ‘public’ ‘private’ ‘protected’. Initialized to ‘public’.

Instance Methods:

a) parentClass - Returns the value of the field parentClass.
b) inheritanceType - Returns the value of the field inheritanceType.
c) parentClass: aClass - Receives a class as the argument and sets the value of the field parentClass accordingly.
d) inheritanceType: aString - Receives a string as the argument and sets the value of the field inheritanceType accordingly.

You may assume that the input to all methods is correct.

Part B – The Multiple Inheritance Model
We do not want to change the existing inheritance mechanism in Squeak, so you will implement the class MyObject, which will support multiple inheritance as described below.

All classes that inherit from MyObject will contain an array of instances of their parent classes. These instances will constitute the sub-objects of the parent classes within the inheriting class. Additionally, at the class level, each class will contain an array of instances of Parent, which will define the multiple parent classes and their inheritance specifiers. You will be required to write a method that will intercept messages sent to objects inheriting from MyObject, and implement a method-search algorithm that takes the specifiers into account. In this assignment we will handle only inheritance of methods and will ignore inheritance of fields.

The following variables and methods must be implemented in MyObject:

Variables:

a) superclasses – a class-instance variable. Important! This is not a class variable. You can read about the difference between class variables and class-instance variables in tutorial 6 or here in chapter 5. This variable will hold an Array of instances of Parent.
b) superInstances – an instance variable that will hold an Array of instances of superclasses.

Methods:
a) Implement the following class method:

\[
\text{subclass: } \text{aSubclassName parentClasses: } \text{anArray instanceVariableNames: } \text{instVarNames classVariableNames: } \text{classVarNames poolDictionaries: } \text{poolDictionaries}\n\text{category: } \text{aCategoryName}
\]

- This method enables the definition of a class that inherits from \text{MyObject} in the regular Squeak inheritance, while also defining the multiple parent classes in the new mechanism. This method is equivalent to the subclass creation method in the language, except for the additional parameter \text{parentClasses: anArray}. The \text{anArray} argument will receive an array of instances of Parent. The method should create the class in the normal Squeak inheritance and then initialize its \text{superclasses} variable using \text{anArray}. The return value is the created class. To initialize the variable look for the method \text{instVarNamed:put:}.

- Your implementation should ensure that all classes inheriting from \text{MyObject} in Squeak’s regular inheritance model will fit the new model. For instance, given a class \text{A} that inherits from \text{MyObject}, if this method is called to define a class \text{B} that inherits from \text{A} with \text{parentclasses: anArray} you must ensure that \text{B} is defined inheriting directly from \text{MyObject} in Squeak’s regular inheritance, and \text{B}’s superclass array is initialized to hold \text{A} with the specifier ‘public’ in the first place, and then the other parents in \text{anArray}.

- Notice that \text{anArray} can be an empty array; and that the parent classes can also be classes that do not inherit from \text{MyObject}. Assume that the input is correct, that the classes do not appear more than once in the array, and that there will not be circular inheritance. Likewise, all classes in \text{anArray} can be instantiated using a call to \text{new}, without an exception being thrown.

b) Override the following method, which is first defined in \text{Class}. Do not change the code in \text{Class}!

\[
\text{subclass: } \text{aSubclassName instanceVariableNames: } \text{instVarNames classVariableNames: } \text{classVarNames poolDictionaries: } \text{poolDictionaries}\n\text{category: } \text{aCategoryName}
\]

- Your implementation should ensure that all classes inheriting from \text{MyObject} in Squeak’s regular inheritance model will fit the new model. For instance, if this method is called to define a class \text{A} that inherits from \text{MyObject}, you must ensure that \text{A} is initialized with an empty array of superclasses. If this method is called to define a class \text{B} that inherits from \text{A} (\text{A} inherits from \text{MyObject}), you must ensure that \text{B} is defined inheriting directly from \text{MyObject} in Squeak’s regular inheritance, and \text{B}’s superclass array is initialized to hold \text{A} with the specifier ‘public’. Use the previous method you defined for this.

c) \text{multiInheritsFrom: aClass} – a class method that receives a class and returns true if \text{aClass} is an ancestor of the current class and false otherwise.

d) \text{superclasses} – a class method that returns the value of the \text{superclasses} variable.
e) **initializeSupers** – an instance method that initializes the superInstances variable to hold an array of instances of the parent classes defined in the superclasses variable. The parent classes should be initialized in the order they are defined in the array.

f) **postInitialize** – an instance method that enables (by overriding in inheriting classes) initialization of the current class that depends on the multiple ancestor classes, because it is called after the parent instances have been initialized. The implementation is given in the file helperFunctions.txt.

g) Override the instance method initialize in MyObject. The implementation is given in helperFunctions.txt.

Note that during initialization, each class in the new model first initializes its parent classes and only afterwards itself, this way the initialization can depend on ancestor classes.

Clarification: In this assignment, ancestor classes are superclasses of the current class, superclasses of the superclasses and so on. If one of the superclasses does not inherit from MyObject, then all classes that this superclass inherits from in Squeak’s regular inheritance are also considered ancestor classes.

**Part C – Method Lookup**

a) Implement the class method classifyInheritedMethod: aSymbol. The method receives the name of a method (as an instance of the class Symbol or a class that inherits from it) and returns one of the following strings:

‘public’ – if the method is considered public in the current class, and there is no ambiguity as defined below.

‘protected’ – if the method is considered protected in the current class, and there is no ambiguity as defined below.

‘private’ – if the method is considered private in the current class, and there is no ambiguity as defined below.

‘ambiguity’ – if a method with this name exists in more than one ancestor of the current class. This can happen in two ways: 1) there are two methods with the same name that were defined for the first time in two different ancestors, or 2) there is an ancestor that defines the method and there is more than one inheritance route between the current class and that ancestor.

‘undefined’ – if the method is not defined in any ancestor class.

‘inaccessible’ – if the method exists in an ancestor class, and there is no ambiguity, but the method cannot be executed by this class because of private inheritance somewhere along the inheritance chain. For example, if the current class inherits from another class which inherits privately from the class defining the method then the method is inaccessible in the current class.
- The method ignores methods defined in the current class, and returns an answer only for inherited methods.
- Make sure to return the strings exactly as they are defined here, the automatic tester relies on this.

b) Implement the **instance method definingInstance: aSymbol**. The method receives the name of a method, (an instance of Symbol or a class that inherits from it) and returns the instance of the ancestor that defines the method, or nil if the method classification is ‘undefined’, ‘inaccessible’, or ‘ambiguity’. This method should search recursively in `superInstances`, and the `superInstances` of the `superInstances` and so on. If the method is defined in a class that doesn’t inherit from `MyObject`, return the superInstance that understands the method.
- The method ignores methods defined in the current class, and returns an answer only for inherited methods.
- For example, let’s look at the following structure: C inherits from B that inherits from A (all inheritance is ‘public’). The method `foo` is defined in A, and the method `bar` in B. In the new model, an instance of C holds an array, `superInstances`, of size 1 that holds an instance of B, and this instance of B holds an array, `superInstances`, of size 1 that holds an instance of A. If we execute the following lines:

```plaintext
c := C new.
x := c definingInstance: #foo.
y := c definingInstance: #bar.
z := c definingInstance: #aaa.
```

Then x will hold the instance of A, y will hold the instance of B, (these are sub objects of c) and z will hold nil.

c) Override the **instance method doesNotUnderstand: aMessage**. This method is called when a message is sent to an object that does not know the method being invoked. Because the new mechanism implemented in the assignment is not known by Squeak, this method will be called every time a message is sent to execute a method not defined in the class itself. In your implementation you must search for the method in the multiple inheritance you created. After the search you must send the method to the correct superclass if possible, taking into account whether the sending class is allowed to execute the method according to the inheritance specifiers.
- You may assume that methods with names understood by `MyObject` will not be added to classes in the model.

Your implementation should do the following (see notes at the end):

1. Find the class that sent this message. To do so use: `thisContext client class`.
2. If a relevant method was found (it’s classification is ‘public’, ‘private’ or ‘protected’) decide whether the sender is allowed to execute this method in the receiver.
a. If so, send the message to the instance of the sub-object that defines the method.
   Use the method `definingInstance: for this`.

3. If a relevant method was not found ('inaccessible', 'undefined', 'ambiguity'), or was found but the sender cannot execute the method, throw an exception using the `class` method:

   `throwSender: senderName fails: methodName inClass: receiverName because: reason`

   The method is given in the file `helperFunctions.txt`. An explanation for the arguments is given there.

   - **Who is allowed to execute a method**: Class B sends a message to class A. if the method is considered private in A then B can execute it only if B==A. If the method is protected in A then B can execute it only if B==A or B inherits from A. If the method is considered public in A then B can execute it.
   - **Important!** You don’t need `thisContext` for anything else in the assignment. However, if you decide to try to do something else with it first file-out your code. A bad access to `thisContext` will cause the VM to crash, and you will lose any work you didn’t export.

### Useful tips

- You may find the solution easier if you consider the method search as a graph search.
- Before you start work, it is recommended that you go over tutorial 6, specifically the parts explaining the 5-level model.
- **Class Methods and Class-instance variables** are defined the same way as instance methods, just click on the “class” button in the system browser under the second column.
- One of the objectives of this assignment is to allow you to understand the abilities of the Squeak language. A large part of the solution is searching for classes and methods that will help you in your solution. Use the Squeak search bar, or try to search for classes by category in the System Browser.
- You are expected to try to figure out for yourselves how to solve the assignment. Questions of the sort “how can we do this” will not be answered unless you show that you have already tried and failed to find the answer.
- You may not add classes other than the ones you were required to add for the assignment.
- Do not change code in classes other the ones you were specifically told to.
- Do not override or change any methods whose name starts with “basic”.
- You can add fields and methods as you wish.
- You may not print to Transcript. If you choose to print to it for testing purposes, make sure to remove the prints before handing in your assignment.
- Do not check types in method arguments unless asked to specifically.
- Comment any non-trivial code you write. Add a short description for every method you write. There’s no need to go overboard with the documentation.
• If you feel stuck – Google is your friend. If you can’t find what you need when searching for Squeak, try searching for Smalltalk (There’s more information about Smalltalk than about Squeak, which is a kind of Smalltalk).

Submission Details

• Requests for postponement, for any reason, must be sent to the TA in charge of the course (Nurit). Note that the course has a late-submission policy, meaning you can submit your assignment late without a postponement approval. Details can be found in the course site under “General Info”.

• The assignment must be submitted electronically. (Save the submission confirmation).

• Your submission should consist of a zip file with the name format: OOP2_<ID1>_ID2>.zip containing:
  i. A file called readme.txt with the name, id and email of each of the submitters.
  ii. OOP3.st (Your code)

• Do not submit the provided class files, and do not submit any tests. Do not submit any classes you created that inherit from MyObject. These may cause problems in the tester.

• Points will be deducted for not conforming to the submission requirements (rar instead of zip, extra files, extra classes, a readme file with the wrong name, etc.)

• Good Luck!