Advanced Java Programming

Part 1: class defined attributes vs. locals, scoping rules, shadowing attributes, relationships between classes, multiplicity

James Tam

Attributes Vs. Locals

- **Attributes**
  - Declared inside a class definition but outside the body of a method

```
public class Person {
    private String [] childrenName = new String[10];
    private int age;
}
```

- **Locals**
  - Declared inside the body of a method

```
public class Person {
    public nameFamily() {
        int i;
        Scanner in = new Scanner(System.in);
}
```

James Tam

Scope Of Attributes Vs. Locals

- **New term**: Scope is the location where an identifier (attribute, local, method) may be accessed
  - Scope of attributes (and methods): anywhere inside the class definition
  - Scope of locals: after the local has been declared until the end of closing brace (e.g., end of method body)
- Example:

```
public class Person {
    private String [] childrenName = new String[10];
    private int age;

    public nameFamily() {
        int i;
        for (i = 0; i < 10; i++) {
            childrenName[i] = in.nextLine();
        }
    }
}
```

Local (method scope)

Attribute (class scope)

James Tam

When To Use: Attributes

- Typically there is a separate attribute for each instance of a class and it lasts (available for) for the life of the object.

```
public class Person
{
    private String [] childrenName = new String[10];
    private int age;
    /*
      For each person it's logical to track the age and
      the names any offspring.
    */
}
```

Q: Life of an object?

James Tam

When To Use: Locals

- Local variables: temporary information that will only be used inside a method

```
public nameFamily()
{
    int i;
    Scanner in = new Scanner(System.in);
    for (i = 0; i < 10; i++)
    {
        childrenName[i] = in.nextLine();
    }
}
```

Scope of 'i' (int)

Scope of 'in' (Scanner)

- Q: Does it make sense for every 'Person' to have an 'i' and a 'in' attribute?

James Tam

A Common Language-Based Convention

- Variables that are used as loop controls are sometimes declared as local only to the loop.
- Example:

```
for (int j = 1; j <= 4; j++)
{
   System.out.print(j + " ");  // In scope
}
// Error: Not in scope
// j = 0;
```

James Tam

Scoping Rules

- Rules of access
  1. Look for a local (variable or constant)
  2. Look for an attribute
- General example

```
public class Person
{

    public void method()
    {

        x = 12;
    }
}
```

Second: look for the definition of the class e.g., "private int x;"

First: look for the definition of a local identifier e.g., "int x;"

Reference to an identifier

James Tam

Scoping Rules: Example

```
public class C
{
    private int x;
    public void m()
    {
        int y;

        x = 1;
        y = 2;
    }
}
```

James Tam

Scoping Rules: Example

Name of the folder containing the complete example:
`1simpleScope`

```
public class C
{
    private int x;
    public void m()
    {
        int y;

        x = 1;
        y = 2;
    }
}
```

James Tam

New Term: Shadowing

- The name of a local matches the name of an attribute.
- Because of scoping rules the local identifier will 'hide' (shadow) access to the attribute.
- This is a common logic error! (You typically do it unintentionally).
- **Name of the folder containing the complete example**:
  `2scopeWithShadowing`

```
public class Person {
    private int age = -1;
    public Person(int newAge) {
        int age;  // Shadows/hides attribute
        age = newAge;
    }
    public void setAge(int age) { // Shadow/hide attribute
        age = age;
    }
}
Person aPerson = new Person(0);  // age is still -1
aPerson.setAge(18);              // age is still -1
```

James Tam

New Term: Messaging Passing

- Invoking the methods of another class.

```
class Driver
{
     main ()
     {
       Game aGame = new Game();
       aGame.start();
     }
}
```

Run method

Run method

```
class Game
{
     Game()
     {
          :
     }
     start()
     {
          :
     }
}
```

James Tam

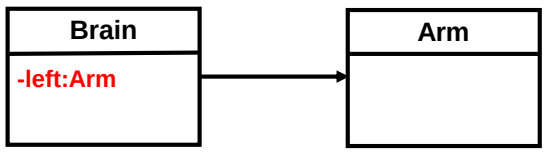
Relationships Between Classes

- **New term**: **Association relation** (“*has-a*”) exists between classes if an instance of one class is an attribute of another class.
- Unidirectional association relation:
  - **Example**:

```
Public class Brain
{
     private Arm left;
     ...
}
```

```
public class Arm
{
     ...
}
```

  - **UML**:



James Tam

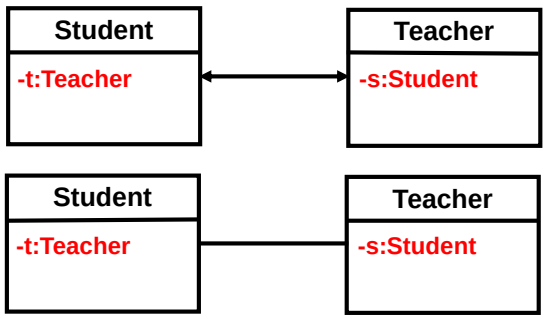
Relationships Between Classes (2)

- Bidirectional association relation:
  - **Example**:

```
public class Student
{
      private Teacher t;
}
```

```
public class Teacher
{
      private Student s;
}
```

  - **UML**:



James Tam

Associations And Message Passing

- Having an association between classes allows **messages to be sent** from one object to another (objects of one class can call the methods of another class).

```
public class Car
{
    private Engine anEngine;
    private [] Lights carLights;
    ...
    public start()
    {
       anEngine.ignite();
       carLights[0].turnOn();
       ...
    }
}
```

```
public class Engine
{
    public boolean ignite() { ..
    }
}
public class Lights
{
    private boolean isOn;
    public void turnOn() {
        isOn = true;}
}
```

- Unidirectional: messages can be sent from car to engine or car to lights but not vice versa.

James Tam

Extra Exercise (Advanced)

- How do we ensure that:
  - A particular instance of one class refers to a particular instance of a second class?

  And
  - That instance of the second class refers to the previously referred to instance of the first class?
- **Name of the folder containing the complete example**: `3relationships`
- What is wrong with the code?
- How can it be fixed?

James Tam

The `Driver` Class

```
public class Driver
{
    public static void main(String [] args)
    {
        Student s = new Student();
        System.out.println("<< DEBUG: This message will never
                            appear >>");
    }
}
```

James Tam

Class Student & Teacher

```
public class Student {
    private Teacher t;
    public Student() {
        t = new Teacher();
    }
}

public class Teacher {
    private Student s;
    public Teacher() {
        s = new Student();
    }
}
```

- JT's hint: similar to the "chicken and the egg problem"!

James Tam

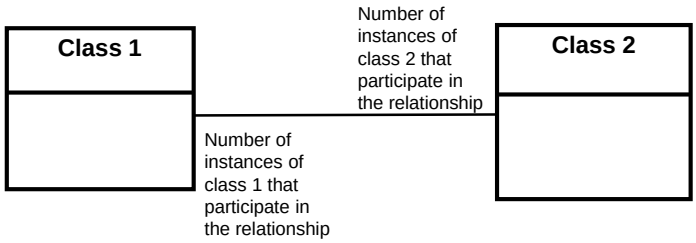
New Term: Multiplicity

- It indicates the number of instances that participate in a relationship

Multiplicity	Description
1	Exactly one instance
n	Exactly "n" instances {n: a positive integer}
n..m	Any number of instances in the inclusive range from "n" to "m" {n, m: positive integers}
*	Any number of instances possible

James Tam

Multiplicity In UML Class Diagrams



James Tam

Why Represent A Program In Diagrammatic Form (UML)?

- Images are better than text for showing structural relations.

Text

Jane is Jim's boss.

Jim is Joe's boss.

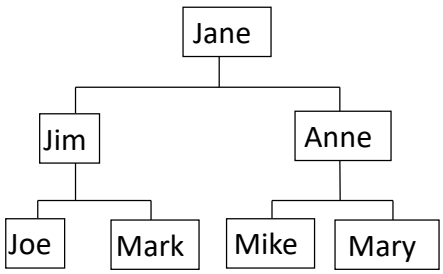
Anne works for Jane.

Mark works for Jim

Anne is Mary's boss.

Anne is Mike's boss.

Structure diagram



- UML can show relationships between classes at a glance

James Tam

Relationships Between Classes

- Design rule of thumb.
- It can be convenient to create a relationship between classes (allow methods to be invoked/messages to be passed).
- But unless it is necessary for a relationship to exist between classes do not create one.
- That's because each time a method can be invoked there is the potential that the object whose method is called can be put into an invalid state (similar to avoiding the use of global variables to reduce logic errors).

James Tam

New Terminology/Definitions

- Scope
- Shadowing
- Message passing
- Association relation (bidirectional, unidirectional)

James Tam

After This Section You Should Now Know

- What is meant by scope
- Scoping rules for attributes, methods and locals
  - Design issues
    - When should something be declared as local vs. an attribute
- The hierarchy of scoping rules
  - How locals can shadow attributes
- What is meant by message passing
- What is an association, how do directed and non-directed associations differ, how to represent associations and multiplicity in UML
- What is multiplicity and what are kinds of multiplicity relationships exist
- Design and technical issues related to association relations

James Tam

Copyright Notification

- "Unless otherwise indicated, all images in this presentation are used with permission from Microsoft."

slide 24

James Tam