CSC 142

Object based programming in Java

[Reading: chapter 2-3]

What we have seen so far

- A class is made of
  - instance fields: attributes of the class objects
  - instance methods: behavior of the class objects (=what objects can do).
- instance methods+fields=instance members
- Some members are public (available to anyone=interface), some are private (available inside of the class only=implementation details)
- To create an object of the class type, use

```
ClassName myObject = new ClassName();
//creates an instance of ClassName
```


Another example

```
import java.awt.Color; // Color class
import uwcse.graphics.*;

public class SomeGraphics{
   // instance fields
   private GWindow window=new GWindow();
   private Oval circle1=new Oval();
   private Oval circle2=new Oval();
   // constructor
   public SomeGraphics(){
    // circle1 is blue, circle2 red
    circle1.setColor(Color.blue);
    circle2.setColor(Color.red);
    // move circle2 to the right by 80 pixels
    circle2.moveBy(80,0);
    window.add(circle1);
    window.add(circle2);
  }
}
```


java.awt

- A (huge) package to do graphics (awt stands for abstract windowing toolkit)
- Color is a class that contains some predefined colors (you can also define your own colors, check the class documentation)
- We could have also written to import the full package
  - import java.awt.*;
  - Note: With the above, we get access to all the classes in java.awt, but not the content of any package in java.awt

Initializing instance fields

- Initializing means giving a value to something for the first time.
- Instance fields can be directly initialized when declared

```
private Oval circle1 = new Oval();
```

- If not, an instance field referring to an object is automatically initialized to null

```
private GWindow w; // w is null
```

- Using a null reference to call a method is an error.

```
w.doRepaint(); // error!
```

Note: doRepaint means redisplay the window

- Before using an instance field, make sure that it refers to an actual object.

references and assignment (1)

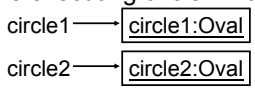
- Consider

```
public SomeGraphics(){
  // circle1 is blue, circle2 red
  circle1.setColor(Color.blue);
  circle2.setColor(Color.red);
  // move circle2 to the right by 80 pixels
  circle2.move(80,0);
  circle2 = circle1;
  window.add(circle1);
  window.add(circle2);
}
```

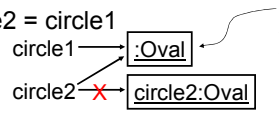
- What happens?

references and assignment (2)

- Before executing circle2 = circle1



- circle2 = circle1



Note: if 2 or more references refer to the same object, drop the object reference in the UML notation

- The object that circle2 referred to can't be accessed anymore. The memory it uses is freed by the JVM garbage collector.
- Only 1 blue circle appears in window

Swapping colors (1)

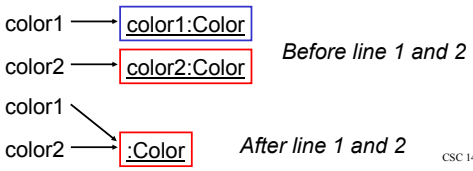
- Add an instance method to the SomeGraphics class to swap the colors of the 2 circles
  - method swapColors: make it public
  - swapColors doesn't require any input. Write swapColors()
  - swapColors doesn't give back a result. Write void to the left of the method name.

```
public void swapColors()
{
  // what should we write?
}
```


Swapping colors (2)

- Is this OK?

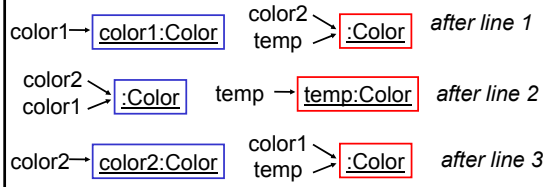
```
Color color1 = circle1.getColor();
Color color2 = circle2.getColor();
color1 = color2; // line 1
color2 = color1; // line 2
circle1.setColor(color1);
circle2.setColor(color2);
```



Swapping colors (3)

- Need a 3rd variable: replace line 1 and 2 with

```
Color temp = color2; // line 1
color2 = color1; // line 2
color1 = temp; // line 3
```



Swapping colors: code

```
public class SomeGraphics{
  // instance fields, constructor as before...
  public void swapColors()
  {
    // colors of the circles
    Color color1=circle1.getColor();
    Color color2=circle2.getColor();
    // swap the colors
    Color temp = color2;
    color2 = color1;
    color1 = temp;
    // repaint with the new colors
    circle1.setColor(color1);
    circle2.setColor(color2);
    window.doRepaint();
  }
}
```

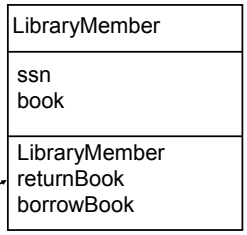
color1, color2 and temp are defined inside of the method. They are called local variables. More on this later.

Designing classes

- Example: a library member
  - The problem (unrealistically simple):
    Library member identified with SSN
    Only one book can be borrowed
    No time limit to return a book
    Book identified by its title
  - Analyze the problem
    nouns=objects, e.g. library member, book, SSN
    verbs=methods, e.g. borrow, return

LibaryMember class

- Two private instance fields
  - ssn
  - book
- Three public methods
  - constructor
  - returnBook
  - borrowBook



The class diagram is incomplete (e.g., no type for the instance fields, no information about the methods)

Instance field: ssn

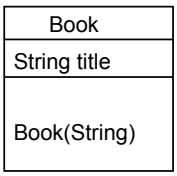
- An integer: int .
- In Java, for efficiency reasons, some types are not object types: boolean for logical variables (true, false), int for integers (e.g., -1, 36,…), double for floating point numbers (e.g., 1.134, 0.23e-11,…), char for characters (e.g. 'a', '!', …)
- write

```
int ssn; // with the instance fields
ssn = 27182813; // e.g. in the constructor
                //  don't use new
```

- More on this later.

Instance field: book

- need a Book class
  - instance field: title of type String (class available in java to manipulate text)
  - instance method: constructor.
    To construct a book, need a String for the title. Pass the String as a parameter to the constructor.



Interlude: String class

- Available in the package java.lang
- Using String objects
  - for String literals, use double quotes

```
"Washington State"
```

  - to create a String, use a String constructor

```
String country = new String("USA");
```

  - how is this different?

```
String country = "USA";
```

  - to get the number of characters in a String

```
int n = state.length(); // n is 5 if
                        // state is "Idaho"
```

- Check the documentation

Book class

```
import java.lang.String;
// unnecessary. java.lang.* is always imported.

public class Book{
  private String title;
  public Book(String bookTitle)
  {
     // use the String constructor
     title = new String(bookTitle);
     // Does title=bookTitle do the same thing?
  }
}
```

- To instantiate the Book class

```
Book someBook = new Book("The grapes of wrath");
```


LibraryMember class constructor

- To construct a LibraryMember, need an ssn
- The constructor needs an input parameter, namely an int.
- Initially, the LibraryMember has not borrowed a Book. book should be null.
- Code (inside the LibraryMember class)

```
public LibraryMember(int someSSN)
{
 ssn = someSSN;
 // Do we need to set book to null?
}
```

Instance method: returnBook (1)

- Input: none
- Output: did the operation succeed? (need to have a book initially). Return a boolean (true if success, false if failure).
- Code (inside the LibraryMember class)

```
public boolean returnBook()
{
  // What do we write?
  ...
```

- Can the LibraryMember return a Book?
  - Yes, if book is not null: set book to null
  - No, if book is null.

Conditionals: a first view

- General structure

```
if (condition)
{
   statement1;
   statement2;
}
else /*can be omitted if no else case*/
{
   statement3;
   statement4;
}
```

condition must be a boolean. It is an error to omit the parentheses

executed if condition is true

executed if condition is false

If there is only one statement after the if or the else, the braces { and } after the if or the else may be omitted.

Equality and relational operators

- All relations are boolean expressions. They evaluate to true or false.
- equal and not equal

```
x == y // is x equal to y?
x != y // is x not equal to y?
```

It is a syntax error to write a condition as x=y instead of x==y

- ordering

```
x < y // is x less than y?
x >=y // is x greater than or equal to y?
x > y // is x greater than y?
x <=y // is x less than or equal to y?
```


What can you compare?

- use ==, !=, <=, >=, <, and > with numbers and characters (e.g., int, double, char). With booleans only == and != are valid.
- **Don't** compare objects with ==, !=, …except when comparing to null. See later.

Instance method: returnBook(2)

- Write within the LibraryMember class

```
public boolean returnBook()
{
  if (book != null) //There is a book
  {
    book=null;
   System.out.println("book returned");
   return true;
  }
  else // There is no book
  {
   System.out.println("Can't return a book");
   System.out.println("You don't have a book");
   return false;
  }
}
```


Console output in Java

- Use the out object in java.lang.System
  - System: a class to access low level system objects and methods (e.g. I/O functionalities)
  - out: PrintStream object. It comes with methods to display text in a console window (i.e., not a graphics window), e.g.
    - `print:` print a String
    - `println:` same as print but add a newline

```
// The following prints Hello, world!
System.out.print("Hello, ");
System.out.print("world!");
// The following prints Hello on one line and
// Good bye on the next line
System.out.println("Hello");
System.out.println("Good bye");
```

return statement

- In a method
  - syntax **`return expression;`**
  - expression must have the same type as the method.
  - transfer the control flow back to the line of code that called this method.

```
public        String      appVersion()
{
  // Send back the current version of the program
  return "This is version 3.1";
}
```

The expression next to the return keyword must be a String

Instance method: borrowBook (1)

- Input: title of the book to borrow
- Output: did the operation succeed? (only one book can be borrowed). Return a boolean (true if success, false if failure).
- Code (inside the LibraryMember class)

```
public boolean borrowBook(String someTitle)
{
  // What do we write?
  ...
```

- Can the LibraryMember borrow a Book?
  - Yes, if book is null. No, if book is not null.

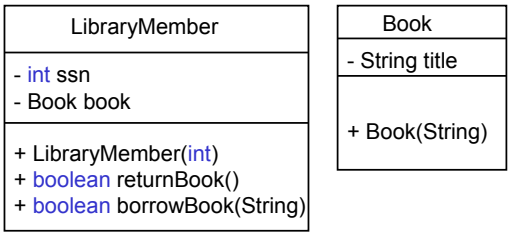
Instance method: borrowBook (2)

- Code (inside the LibraryMember class)

```
public boolean borrowBook(String someTitle){
  if (book==null) {
    // Can borrow the book
    book = new Book(someTitle);
    // Note the use of + to concatenate Strings
    System.out.println("Borrow " + someTitle);
    return true;
  }
  else {
    // Can't borrow the book
    // Note '\n' to print a new line
    System.out.println("You can't borrow "+
     someTitle +"\nYou already have a book");
    return false;
  }
}
```


Putting it all together (1)

- class diagrams: + for public, - for private



Putting it all together (2)

```
public class LibraryMember{
  // id and borrowed book
  private int ssn;
  private Book book;
 // constructor
  public LibraryMember(int someSSN){ssn=someSSN;}
 // other methods (see previous slides)
  public boolean returnBook(){/*code*/}
  public boolean borrowBook(String someTitle)
  {
     // code
  }
}
```


Using the class LibraryMember

```
public class UsingLibraryMember{
 public void testLibraryMember()
 {
  // Borrow and return and see if it works
  LibraryMember lm=new LibraryMember(123456789);
  lm.borrowBook("The great Gatsby");
  if (lm.returnBook())
     System.out.println("It is working");
  if (lm.borrowBook("The old man and the sea"))
     System.out.println("It is still working");
  lm.borrowBook("Learning Java");
  lm.returnBook();
  lm.returnBook();
  lm.borrowBook("Learning Java");
 }
}
```