

CSE1030 – Introduction to Computer Science II

Lecture #2 Introduction to Object Oriented Programming

Goals for Today

- Theory:
 - Learn a little about Object Oriented Programming
- Practical: (Assignment #1!)
 - How to create a Java class
 - What makes a class a Static or Utility Class
 - JavaDocs

CSE1030 2

CSE1030 – Lecture #2

- Intro to Object Oriented Programming
- Elements of a Java Class
- Utility Classes
- JavaDoc
- We're Done!

CSE1030 3

Idea Behind OOP

- Make it easier to develop and maintain large or complex software systems
- Originated in the original Graphical User Interface research projects (complex!)
- Fundamental Ideas:
 - Organise Data and Code into Modules
 - Formalise the way one module interacts with another (We call this the **interface** between the Modules)

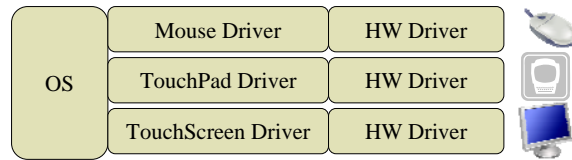


Sketchpad (1963)

CSE1030 4

Object Oriented Programming

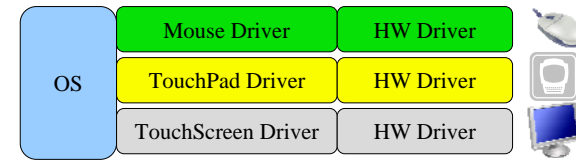
- Object Oriented Programming
 - Example:



CSE1030 5

Object Oriented Programming

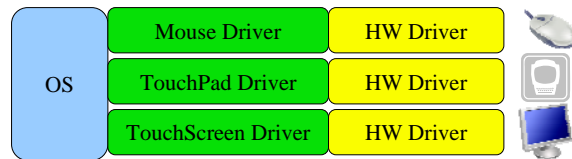
- Object Oriented Programming
 - Is Not:** Task Oriented
 - Is Not:** Library Oriented



CSE1030 6

Object Oriented Programming

- Object Oriented Programming
 - Is:** A Different Way of **Thinking About** the Organisation of the Program and its Data



CSE1030 7

Why OOP?

- Encapsulation
 - Data & Code* in single well-defined location
 - Hide complexity away, on expose a simple API**
- Take Advantage of Inherent Relationships
 - Polymorphism
 - Objects that do similar things are often used similarly
 - Inheritance
 - Many things are "a kind of..." something else

*Code = Software

**API = Application Programming Interface

CSE1030 8

Java and OOP

- Java is an Object Oriented Language
- Big Idea:
In Java, **Everything** is an **Object***
(* almost, we'll talk more about this later)
- And a Java **Class** is how Objects are Defined

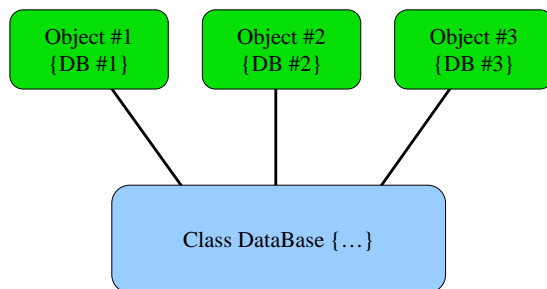
CSE1030 9

Java Classes

- Classes describe Objects ←Important Idea!
(Every Object has a Class)
- Java Class Definition: (we'll come back to this)
 1. **Names** the Class
 2. Describes **How to Construct** an Object of the Class
 3. Stipulates **Who** can use our Objects, and **How**
 4. **Defines the Data** in the Objects (and in the Class)
 5. **Contains all of the Code** pertaining to the Objects

CSE1030 10

How it Normally Looks when Running



CSE1030 11

CSE1030 – Lecture #2

- Intro to Object Oriented Programming
- Elements of a Java Class →
- Utility Classes
- JavaDoc
- Parameters and Parameter Passing
- We're Done!

CSE1030 12

Elements of a Java Class

```
// any needed package statement
// any needed import statements

public class ClassName
{
    // data declarations
    private int i;

    // constructor
    ClassName(){ i = 0; };

    // method definitions
    int getI() { return i; }
    void setI(int pi) {i = pi; }
}
```

CSE1030 13

Elements of a Java Class

```
// any needed package statement
// any needed import statements

1. Name the Class → public class ClassName
2. How to Construct an Object →
3. Who can use, and How →
4. Defines the Data →
5. Contains the Code →

// data declarations
private int i;

// constructor
ClassName(){ i = 0; };

// method definitions
int getI() { return i; }
void setI(int pi) {i = pi; }
}
```

CSE1030 14

Elements of a Java Class - Name

1. Name the Class

- The name should be unique and meaningful
- Class name must be the same as the name of the file, for example this class should be saved in a file called "ClassName.java"

```
// any needed package statement
// any needed import statements

public class ClassName
{
    // data declarations
    private int i;

    // constructor
    ClassName(){ i = 0; };

    // method definitions
    int getI() { return i; }
    void setI(int pi) {i = pi; }
}
```

CSE1030 15

Elements of a Java Class - Constructor

2. How to Construct an Object

- This function is called the **Constructor**
- Its primary purpose is to initialise the data elements (like the variable `i`)
- It must have the same name as the class
- It does not have a "return type", its return type is implicit
- If you do not write a constructor for a class, then one is automatically generated for you
- Consequently, if you do not want people to be able to create objects of this type, you must use an access specifier (e.g., "private")

```
// any needed package statement
// any needed import statements

public class ClassName
{
    // data declarations
    private int i;

    // constructor
    ClassName(){ i = 0; };

    // method definitions
    int getI() { return i; }
    void setI(int pi) {i = pi; }
}
```

CSE1030 16

Elements of a Java Class - Access

3. Who can use, and How

- Access Specifiers control who has access to what parts:
 - public – everybody
 - protected – only related classes (this class, subclass, package*)
 - private – only this class
 - 'none' ~ protected (no sub classes)
- Only 1 public class per file (class with the same name as the file)
- Can be applied to members too:
`private int i;`

```
// any needed package statement
// any needed import statements
public class ClassName
{
    // data declarations
    private int i;

    // constructor
    ClassName(){ i = 0; };

    // method definitions
    int getI() { return i; }
    void setI(int pi) {i = pi; }
}
```

* A **Package** is a group of Classes. Classes without package specifiers go into the **default** package.

CSE1030 17

Elements of a Java Class - Data

4. Defines the Data

- Data within a class is usually made private
- Accessor functions are used to control access to this internal data, examples
- Mutators change data values
- The data members are initialised by the constructor

```
// any needed package statement
// any needed import statements
public class ClassName
{
    // data declarations
    private int i;

    // constructor
    ClassName(){ i = 0; };

    // method definitions
    int getI() { return i; }
    void setI(int pi) {i = pi; }
}
```

CSE1030 18

Elements of a Java Class - Code

5. Contains the Code

- All operations pertaining to the objects of this class should be performed by functions defined right here, in this class.
- This way, everybody can see all of the related data and code in one place.

```
// any needed package statement
// any needed import statements
public class ClassName
{
    // data declarations
    private int i;

    // constructor
    ClassName(){ i = 0; };

    // method definitions
    int getI() { return i; }
    void setI(int pi) {i = pi; }
}
```

CSE1030 19

CSE1030 – Lecture #2

- Intro to Object Oriented Programming
- Elements of a Java Class
- Utility Classes**
- JavaDoc
- We're Done!

CSE1030 20

Definition of a Utility Class

- A Class that contains a common often re-used function (or family of functions)...
- No Objects – usually they are collections of functions
- Examples:
 - java.lang.Math
 - java.lang.System
 - java.util.Collections

CSE1030 21

No Objects?

- Why would we use an Object Oriented Language to write code that doesn't have any objects?
- ANSWER: What if I have a very simple little thing that just doesn't need classes? Like Adding a Couple of Numbers? What code do I actually need?

CSE1030 22

Example: Objects Not Necessary

```
public class DeclareVariables
{
    public static void main(String[] args)
    {
        int A = 10;
        int B = 20;
        int C = A + B;

        System.out.println("The answer is: " + C);
    }
}
```

CSE1030 23

Example: But it's still a class

```
public class DeclareVariables
{
    public static void main(String[] args)
    {
        int A = 10;
        int B = 20;
        int C = A + B;

        System.out.println("The answer is: " + C);
    }
}
```

← Declare the Class

↑ Define a Member: `main()`

CSE1030 24

The `main()` Function

- The main function is where execution of all java programs begins
- All classes can have a main function
 - Even if there are more than one class, each can have it's own main function
 - The only main function that matters is the one in the controlling class – that is the one that will be run
- The main function is labelled static, meaning that an object is not needed to run the main function
 - That's great if we don't want the added complexity of having objects around

CSE1030 25

Problem...

Even though we haven't provided a constructor in our example, Java will automatically create one for us.

So to ensure that nobody creates an object of a class we don't want them to, we have to *disable* the constructor by making it **private**

CSE1030 26

Now we have a Class, but no Objects

```
public class DeclareVariables
{
    private DeclareVariables() {};

    public static void main(String[] args)
    {
        int A = 10;
        int B = 20;
        int C = A + B;

        System.out.println("The answer is: " + C);
    }
}
```

CSE1030 27

Characteristics of Utility Classes

- Want to make this functionality available to others
- Usually Utilities are collections of useful functions, rather than stand-alone programs

CSE1030 28

The AdditionUtility Class

```
public class AdditionUtility
{
    private AdditionUtility() {};

    public static int add(int A, int B)
    {
        return A + B;
    }
}
```

CSE1030 29

Summary: Utility Classes

- Private Constructor
- **All** members, data and code, must be labelled **static**
- Usually does not contain a main() function

CSE1030 30

Example: Using the Utility Class

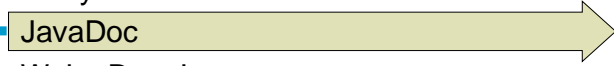
```
class Client
{
    public static void main(String[] args)
    {
        int A = 10;
        int B = 20;

        //    int C = A + B;
        int C = AdditionUtility.add(A, B);

        System.out.println("The answer is: " + C);
    }
}
```

CSE1030 31

CSE1030 – Lecture #2

- Intro to Object Oriented Programming
- Elements of a Java Class
- Utility Classes
- **JavaDoc** 
- We're Done!

CSE1030 32

Intro to Javadoc

- How do we make it easy for other programmers (or even ourselves) to use our classes?
- Users need to know the API
- Javadoc provides semi-automatic generation of API documentation suitable for viewing in a browser

CSE1030 33

Javadoc Comments

```
/**
 * This class defines a function for
 * adding two numbers
 */
public class AdditionUtility
{
    private AdditionUtility() {};

    /**
     * This function adds two numbers.
     */
    public static int add(int A, int B)
    {
        return A + B;
    }
}
```

CSE1030 34

Running javadoc

- `javadoc AdditionUtility.java`
- This command reads the Java source file and generates the API documentation.
- Several files are created, open the one called "index.html" in your web browser

CSE1030 35

Class AdditionUtility

Note! This is an old-style javadoc

java.lang.Object
└─ AdditionUtility

public class AdditionUtility
extends java.lang.Object

This class defines a function for adding two numbers

Method Summary

static int	add(int A, int B)
	This function adds two numbers.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Method Detail

add

public static int add(int A,
int B)

This function adds two numbers.

CSE1030 36

JavaDoc Tags...

- It would be nice to include more information regarding our function
- How about, what the parameters are, what's the return value?

CSE1030 37

Adding Details to `add()`

```
/**
 * This function adds two numbers.
 *
 * @param A A number to add
 * @param B Another Number to add
 * @return The sum, A + B
 */
public static int add(int A, int B)
{
    return A + B;
}
```

CSE1030 38

Method Detail

`add`

```
public static int add(int A,
                    int B)
```

This function adds two numbers.

Parameters:

A - A number to add
B - Another number to add

Returns:

The sum, A + B

CSE1030 39

Preconditions

- Preconditions are instructions made to the users of your function
- You should **always check** the validity of **your function's parameters**
- But **if you have limits** in what you can handle, **tell the user** – use a precondition!

CSE1030 40

Adding javadoc Preconditions

```
/**
 * This function adds two numbers.
 *
 * @param A A number to add
 * @param B Another Number to add
 * @pre. There are no preconditions
 * @return The sum, A + B
 */
public static int add(int A, int B)
{
    return A + B;
}
```

CSE1030 41

Preconditions in JavaDoc

- Preconditions are not directly supported in this version of javadoc
- So we use a custom tag: **@pre.**
- And we run a more complicated javadoc command:

```
javadoc -tag param -tag pre.:a:"Precondition: "  
-tag return AdditionUtility.java
```

CSE1030 42

Method Detail

add

```
public static int add(int A,  
                    int B)
```

This function adds two numbers.

Parameters:

A - A number to add
B - Another number to add

Precondition:

There are no preconditions

Returns:

The sum, A + B

CSE1030 43

Final JavaDoc Notes

- There are other tags:
 - @author, @version, @see, @throws, etc.
- You can use HTML tags in the comments
- More information about defining your own custom tags appears in the online javadoc documentation:
<http://docs.oracle.com/javase/1.4.2/docs/tooldocs/windows/javadoc.html#tag>

CSE1030 44

CSE1030 – Lecture #2

- Intro to Object Oriented Programming
- Elements of a Java Class
- Utility Classes
- JavaDoc
- We're Done!

CSE1030 45

Next topic...

Non-Static Object Features

CSE1030 46