CSPP 511-01:
Introduction to Object-Oriented Programming

Harri Hakula

Ryerson 256, tel. 773-702-8584
hhakula@cs.uchicago.edu

July 10, 2000
Outline

- Wrapper Classes
- Classes
- SS# Example
- Testing
- Everything is a Class in JAVA
Wrapper Classes

JAVA makes a distinction between the primitive types and the class types. For every primitive type there is a corresponding class type, a wrapper class, which makes it possible to convert a value of a primitive type to something "equivalent".

For instance, we cannot say:

\texttt{String a = "1234"; int b = a;}

eventhough our intentions are clear and the functionality is desirable. We can say, however,

\texttt{int b = Integer.valueOf(a).intValue();}
Wrapper Classes Cntd.

or

```java
int b = Integer.parseInt(a);
```

The latter is the preferred form. Unfortunately, this is not supported on all JAVA versions.

We could have set `a` as follows:

```java
String a = Integer.toString(1234);
```
Classes

Every class consists of data items and methods or collectively members of an object. They are sometimes called fields.

Savitch uses instance variables for data items and methods for methods. Occasionally we shall use data members for data items.
Classes Cntd.

Every field can either be public or private. Anything public is accessible from outside the class whereas private members are visible only within the class.

Static members are shared by all instances, whether they are public or private.
SS# Example

Let us consider a US-style social security number. A SS# has three dash-separated fields, with three, two, and four digits, respectively. All together 11 symbols are used to represent the data.

By closer inspection we realize that only the digits count, dashes are simply placeholders. We could automatically save 2/11 symbols per SS#, almost 20% savings.

But is it worth it?
SS# Example: Design

What are the relevant operations? What is data, what are the actions?

Let us consider the actions first:

• SS# must be set, it must be given a value.
• SS# must be inspected. Its value must be queried somehow.
• We must verify that the number is valid.
SS# Example: Design Cntd.

The easy way to store the data is to have a String data member.

How to verify? The SS# must have 11 symbols, the fourth and seventh of which are dashes, every are symbol is a digit, and there probably exists a built-in formula for verifying the digits.

If we assume that the value is set with a String, the first two tests are simple with String class. The third one will be skipped in our design.
SS# Example: Design Cndtd.

What is public, what is private?

Minimal public API consists of two methods:

boolean set(String) Value is set from String. Verification is done automatically and the method returns true if String contains a valid SS#.

String get() The value is returned as a String.

The private API has one method:

boolean verify(String) Check String and return true if it contains a valid SS#.
SS# Example: Design Cntd.

Why is verify private? If it were public, someone could verify that a String is a valid SS# and then at some later point accidentally modify it. Here we ensure that only valid SS#s exist.

The data member which contains the data is private, of course.
SS# Example: Implementation

Class SSID: The public API:

```java
public boolean set(String ssid) {
    if(!verify(ssid)) return false;
    ssid_ = ssid;
    return true;
}

public String get() { return ssid_; }
```

where `ssid_` is the private data member of type String.
SS# Example: Implementation Cntd.

Class SSID: The private API: First we test the simple things.

```java
private boolean verify(String ssid) {
    if( ssid.length() != 11 ) return false;
    if( (ssid.charAt(FIRST_DASH) != '-') ||
        (ssid.charAt(SECOND_DASH) != '-') )
        return false;
    return false;
}
```

where constants are used to indicated fixed locations.
SS# Example: Implementation Cntd.

private boolean verify(String ssid): Only digits allowed.

    for(int i = 0; i < FIRST_DASH; ++i)
        if(!Character.isDigit(ssid.charAt(i)))
            return false;
    for(int i = FIRST_DASH + 1;
        i < SECOND_DASH; ++i)
        if(!Character.isDigit(ssid.charAt(i)))
            return false;
    for(int i = SECOND_DASH + 1;
        i < ssid.length(); ++i)
        if(!Character.isDigit(ssid.charAt(i)))
            return false;
SS# Example: Implementation Cntd.

private boolean verify(String ssid): If all conditions are met, return true.

    return true;

}
Testing

We should be able to test our classes with minimal effort. In JAVA there is a mechanism for this. Every class can have its own main-method. Thus, every class can be compiled separately and tested separately (or with the ones it depend on). Consider:

```java
public static void main(String args[]) {
    SSID ssid = new SSID();
    if( ssid.set("111-34-1234") )
        System.out.println( ssid.get() );
    else
        System.out.println( "Not OK." );
}
```
Testing Cntd.

It is essential to test every method in the API. It is a good practice to include the main per class, just so that the reader will get a feel of its use.

For more complicated testing, one should provide a "driver program", which demonstrates the features of the class. The driver may have additional support routines that are not necessary in the actual class that we are testing.
Analysis

SSID uses built-in components, i.e., the String class. The public and private APIs are cleanly separated.

SSID is not space-efficient, 4 bytes (2 per char) are wasted. The state of the object after an invalid assignment does not reflect that this happened.
SS# Example: Improved Version

(The source code is not shown here.)

Often we can trade space with computational efficiency. In this case, we can filter out the dashes and add them only when needed. The total savings are of the order $n \times 4$ bytes, where $n$ is the number of stored SS#s. These savings can be even doubled by using `byte[] ssid_` instead of a String-based implementation.

`set` and `get` are now much more complex, since compression/decompression must be computed every time these methods are invoked.
Everything is a Class in JAVA

In JAVA classes are used to express concepts. There are no methods or variables outside the scope of at least one class.

With static methods we can write classes that provide services for other classes. Wrapper classes are one example of this design. System.out is an example of a data member which is published by its owner, the System class.