Lecture 20

- Covers
  - Information hiding and encapsulation
  - Access modifiers
  - Class interfaces
  - javadoc
- Reading: Savitch 4.2, Appendix 9

Lecture overview

- Access Mode and Encapsulation
- Information Hiding
- The javadoc Utility (to generate documentation on classes)

- Access modifiers and encapsulation

Access modifiers

- In defining an attribute or a method, we can use the following access modifiers to control how the attribute or the method can be accessed by other classes
  - public
  - protected
  - private

Access modes

- These access modifiers give rise to four access modes
  - public
  - package (or default access mode)
  - protected
  - private

Access modifiers / modes

Mode	Modifier	Accessible to
public	*public*	Every class
protected	*protected*	Subclasses and classes in the same package
package	NONE	Classes in the same package
private	*private*	The current class only

Encapsulation issue

- Access modifiers are closely related to the issue of encapsulation
- Encapsulation is often described as the "mechanism to put methods and attributes together"
  - This explanation fails to explain the significance or purpose of encapsulation
- The purpose of encapsulation is to enable the object to exercise proper control over its state
- In particular, to protect the state's integrity

Example - digital clock

- Model a digital clock (that has hours and minutes)
- Want hours between 0 and 23, and minutes between 0 and 59
- Should write the constructors and mutators carefully to enforce the valid ranges

A design

Attributes: hours, minutes

Constructor: no argument; set hours and minutes to 0

Mutator methods (methods that change the state of an object as defined by its attributes):

```
setHours(int hrs)
setMinutes(int mns)
tick( )  // a minute has passed
```

Accessor methods (considered later)

A (wrong) choice of access modes

- Suppose we choose to let the attributes have public access mode

```
public class DigitalClock
{
    public int hours;
    public int minutes;

    public DigitalClock( )
    {
        hours = 0;
        minutes = 0;
    }
```

```
public void setHours(int hrs)
{
        if (0 < = hrs && hrs <= 23)
        {
                hours = hrs;
        }
        else
        {
                hours = 0;
        }
}

public void setMinutes(int mns)
{
        // similar
}
```

```
    public void tick( )
    {
        minutes ++;
        if (minutes == 60)
        {
            minutes = 0;
            hours ++;
        }
        if (hours == 24)
        {
            hours = 0;
        }
    }

    public String toString( ){  }
    // other accessor methods
}
```

Consequences

- Despite our effort to keep hours and minutes in the valid ranges, other classes can easily set them to invalid values
- For example, the class DigitalClockTest can do this with the statement

```
clock.hours = -100;
```

Consequences

- The previous violation is a consequence of the public access mode
- Recall that public attributes can be directly accessed by other classes
- Hence class DigitalClockTest can directly access and change hours and minutes at will

Encapsulation

- Only the private access mode offers proper protection
- In other words, a DigitalClock object has proper control over its state (in particular, the state is ensured to be valid)
- We say that objects are well-encapsulated when their details (particularly instance attributes) are hidden in this way - access and change must be through the class interface (its public operations)

Accessor methods

- We may provide methods to make the values of hours and minutes available (as read-only properties) to other classes
- They are usually known as accessor methods
- The use of accessor methods
  - Enhances the usefulness of the class
  - Will not do any harm - the principle of encapsulation is still enforced
- The name of accessor methods usually starts with the word get, so frequently accessor methods are referred to as get methods

Code for accessor methods

```
public int getHours( )
{
        return hours;
}

public int getMinutes( )
{
        return minutes;
}
```

Mutator methods

- Sometimes we have a legitimate reason for allowing a user of our class to change the value of an attribute
- The name of mutator methods usually starts with the word set, so frequently mutator methods are referred to as set methods

General rule

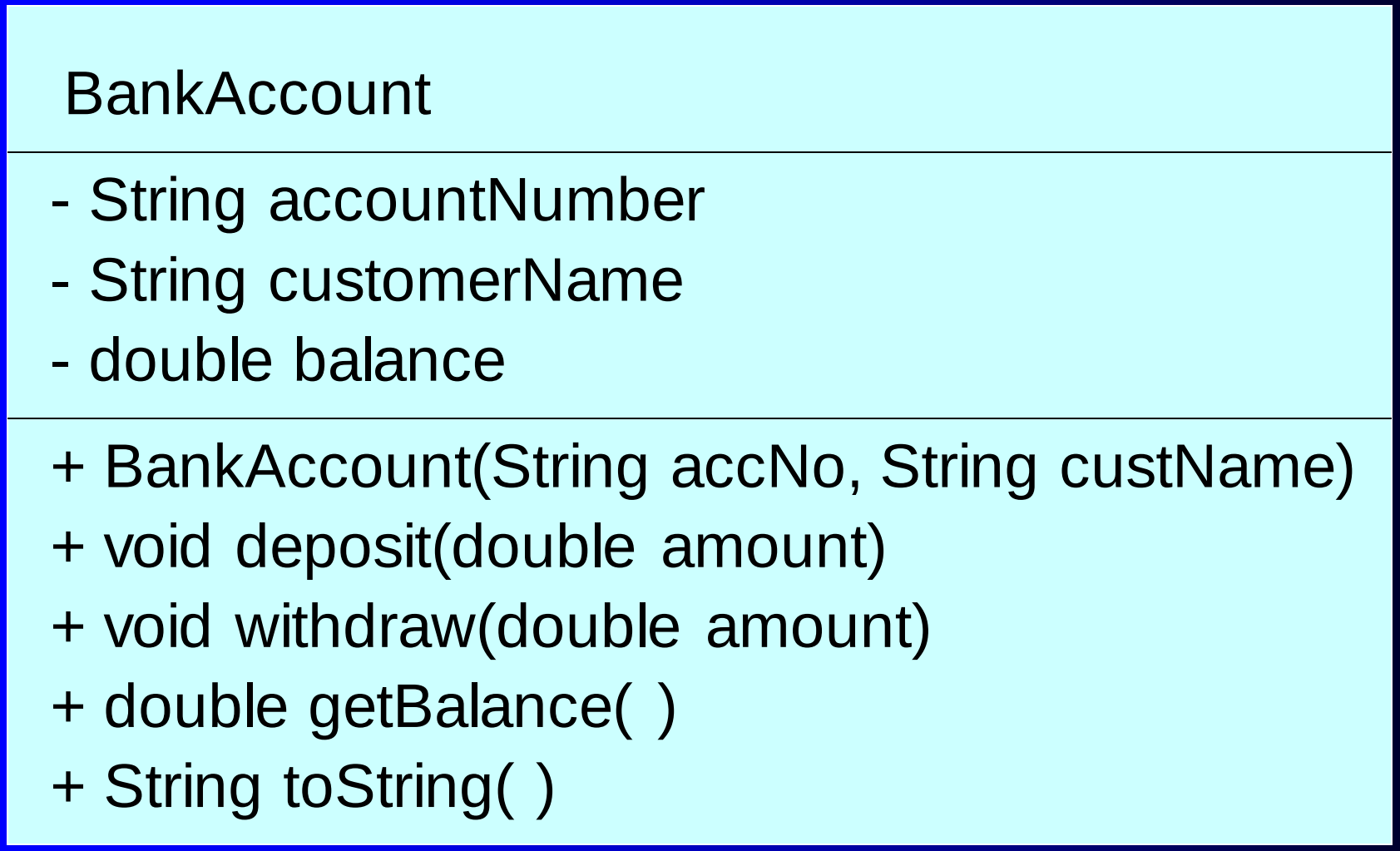
- To enforce the principle of encapsulation
  - Chose private access mode for attributes
  - Provide accessor methods to make the attributes available in the read-only mode to other classes

Access modifiers and UML

- In UML Class diagrams, the access modifier can be specified for each attribute and method
  - + indicates public
  - - indicates private

- If no access modifiers are specified, one would assume that attributes are private and methods are public

Class diagram



General rule

- To enforce the principle of encapsulation
  - Chose private access mode for attributes
  - Provide accessor methods to make the attributes available in the read-only mode to other classes

- Information hiding

Encapsulation (and information hiding)

- Encapsulation
  - A form of information hiding
  - Hide the details of a class and provide an interface to the class which controls access to the object

Information hiding

- Is a means to
  - Reduce the "cognitive load" on a programmer
- Includes
  - Designing classes so they can be used without needing to understand how they are programmed

Preconditions of methods

- When the details of a method are hidden, a user of that method needs to know how to use it
- One approach is to describe (implicitly or explicitly) the preconditions and postconditions of a method
- Preconditions
  - What must be true for the method to function correctly

Postconditions of methods

- Postconditions
  - What holds true after the method has executed
  - For example, what result a return value holds
  - Postconditions describe the effect of calling a method

- Javadoc (to generate documentation on classes)

Terminology

- API – Application Programming Interface
  - Specifies the interface of a Java class
  - Used to refer to the Java class libraries

- javadoc – program that extracts comments out of Java source files and creates an HTML file with the class interface information

javadoc

- See Appendix 9 in Savitch
- Comments start with /** and end with */
- Place comments directly before each class and each method
- The first sentence of method comments should summarise the purpose of the method. Further sentences can give more details as to its use.

javadoc

- In method comments, we can specifically comment parameters
- For example

```
/**
   Sets the minutes field to a new value.
   If minutes is not between 0
   and 59 then it sets minutes to 0.
   @param mns the new value for minutes
   */
   public void setMinutes(int mns)
```

javadoc

- In method comments, we can specifically comment return values
- For example

```
/**
Returns a String representation of the
current state of the digital clock.
@return The String representation of the time
*/
public String toString( )
```

javadoc

- You can run javadoc on a single class or a whole package (explained later)
- On a single class

  javadoc MyClass.java
- Creates a file

  MyClass.html
- The HTML documentation file can be viewed with a web browser such as Netscape, Konqueror or Internet Explorer

** Refer to DigitalClock.java and DigitalClock.html handouts*

Next lecture

- Objects and references
- Parameter passing with class parameters