

3 Object interaction

Creating cooperating objects

Main concepts to be covered

- Abstraction and modularisation
- Object references
- Object creation
- Class diagrams and Object diagrams
- Object copying - shallow and deep copy
- Method calling - internal and external
- Object collections - lists
- Class libraries

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 3 2

A digital clock

11:03

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 3 3

Abstraction and modularization

- **Abstraction** is the ability to ignore details of parts to focus attention on a higher level of a problem.
- **Modularization** is the process of dividing a whole into well-defined parts, which can be built and examined separately, and which interact in well-defined ways.

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 3 4

Modularizing the clock display

11:03

One four-digit display?

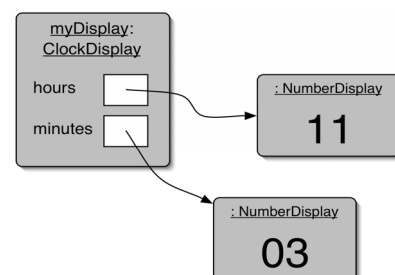
Or two two-digit displays?

11 03

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 3 5

Object diagram



Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 3 6

Implementation - NumberDisplay

```
public class NumberDisplay
{
    private int limit;
    private int value;

    Constructor and
    methods omitted.
}
```

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 3 7

Implementation - ClockDisplay

```
public class ClockDisplay objects
{
    private NumberDisplay hours;
    private NumberDisplay minutes;

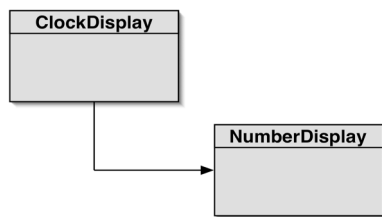
    Constructor and
    methods omitted.
}
```

classes define data types

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 3 8

Class diagram

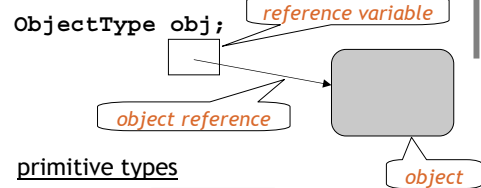


Object oriented programming, DAT042, D2, 11/12, lp 1

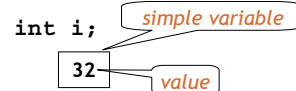
Lecture 3 9

Primitive types vs. object types

object types



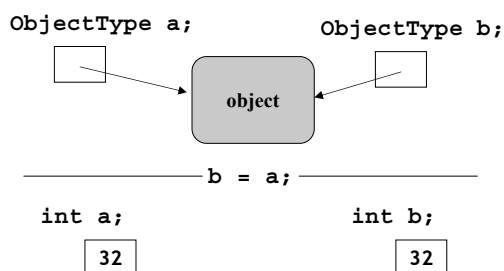
primitive types



Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 3 10

Primitive types vs. object types

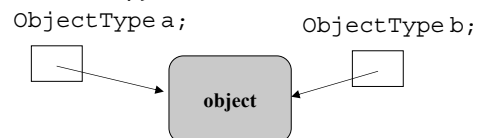


Object oriented programming, DAT042, D2, 11/12, lp 1

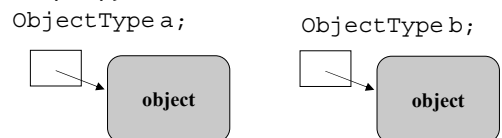
Lecture 3 11

Shallow copy vs. Deep copy

Shallow copy



Deep copy



Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 3 12

Source code: NumberDisplay

```
public NumberDisplay(int rollOverLimit)
{
    limit = rollOverLimit;
    value = 0;
}

public void increment()
{
    value = (value + 1) % limit;
}
```

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 3 13

Source code: NumberDisplay

```
public String getDisplayValue()
{
    if(value < 10) {
        return "0" + value;
    }
    else {
        return "" + value;
    }
}
```

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 3 14

Objects creating objects

```
public class ClockDisplay
{
    private NumberDisplay hours;
    private NumberDisplay minutes;
    private String displayString;

    public ClockDisplay()
    {
        hours = new NumberDisplay(24);
        minutes = new NumberDisplay(60);
        updateDisplay();
    }
}
```

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 3 15

Objects creating objects

in class NumberDisplay:

```
public NumberDisplay(int rollOverLimit);
```

formal parameter

in class ClockDisplay:

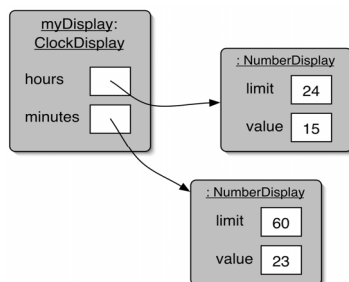
```
hours = new NumberDisplay(24);
```

actual parameter

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 3 16

ClockDisplay object diagram



Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 3 17

Source code: ClockDisplay a private internal method ("a help function")

```
/**
 * Update the internal string that
 * represents the display.
 */
private void updateDisplay()
{
    displayString =
        hours.getDisplayValue() + ":" +
        minutes.getDisplayValue();
}
```

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 3 18

Method calling

```
public void timeTick()
{
    minutes.increment();
    if (minutes.getValue() == 0) {
        // it just rolled over!
        hours.increment();
    }
    updateDisplay();
}
```

external method calls
(objects of class NumberDisplay)

internal method call
(timeTick and updateDisplay
are methods in class ClockDisplay)

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 3 19

Method calling (2)

- internal method calls
 `updateDisplay();`
 `methodName (actual parameters);`
- external method calls
 `minutes.increment();`
 `object . methodName (actual parameters);`

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 3 20

Concepts

- abstraction
- modularization
- classes define types
- class diagram
- object diagram
- object references
- primitive types
- object types
- object creation
- overloading
- internal/external method call

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 3 21

Grouping objects

Collections of objects

Main concepts to be covered

- Class libraries - packages
- Collections of objects - Array lists
- Generic types

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 3 23

The requirement to group objects

- Many applications involve collections of objects:
 - Personal organizers.
 - Library catalogs.
 - Student-record system.
- The number of items to be stored varies.
 - Items added.
 - Items deleted.

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 3 24

A personal notebook

- Notes may be stored.
- Individual notes can be viewed.
- There is no limit to the number of notes.
- It will tell how many notes are stored.
- Explore the *notebook1* project.

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 3 25

Class libraries

- Collections of useful classes.
- We don't have to write everything from scratch.
- Java calls its libraries, *packages*.
- Grouping objects is a recurring requirement
 - The `java.util` package contains classes for various kinds of object collections.

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 3 26

```
import java.util.ArrayList;

/**
 * ...
 */
public class Notebook
{
    // Storage for an arbitrary number of notes.
    private ArrayList<String> notes;

    /**
     * Perform any initialization required for the
     * notebook.
     */
    public Notebook()
    {
        notes = new ArrayList<String>();
    }

    ...
}
```

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 3 27

Collections of Objects

- We specify:
 - the type (kind) of collection: `ArrayList`
 - the type of objects it will contain: `String`
- We say, “`ArrayList of String`”.
- `ArrayList` implements list functionality:
 - `add`, `get`, `size`, etc.

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 3 28

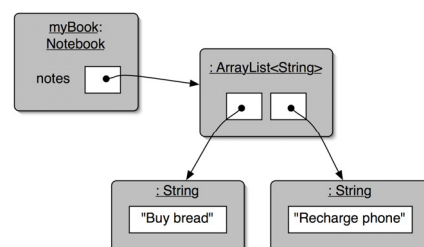
Generic classes

- Collections are known as *parameterized* or *generic* types.
- The type parameter says what we want a list of:
 - `ArrayList<Person>`
 - `ArrayList<TicketMachine>`
 - etc.
- So if `T` is a type, then `ArrayList<T>` is a type as well.

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 3 29

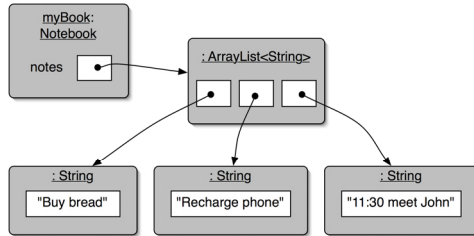
Object structures with collections



Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 3 30

Adding a third note



Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 3 31

Features of the collection

- It increases its capacity as necessary.
- It keeps a private count (`size()` accessor).
- It keeps the objects in order.
- Details of how all this is done are hidden.
 - Does that matter? Does not knowing how prevent us from using it?

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 3 32

Using the collection

```

public class Notebook
{
    private ArrayList<String> notes;
    ...

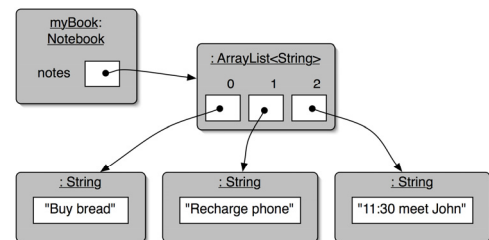
    public void storeNote(String note)
    {
        notes.add(note); ← Adding a new note
    }

    public int numberOfNotes()
    {
        return notes.size(); ← Returning the number of notes (delegation)
    }
    ...
}
    
```

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 3 33

Index numbering



Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 3 34

Retrieving an object

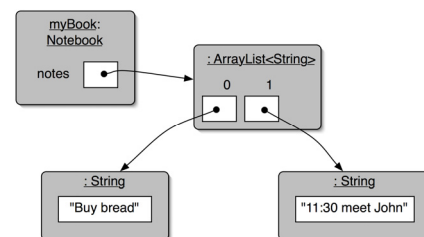
```

public void showNote(int noteNumber)
{
    ← Index validity checks
    if(noteNumber < 0) {
        // This is not a valid note number.
    }
    else if(noteNumber < numberOfNotes()) {
        System.out.println(notes.get(noteNumber));
    }
    else {
        // This is not a valid note number.
    }
}
    ← Retrieve and print the note
    
```

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 3 35

Removal may affect numbering



Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 3 36

Review

- Collections allow an arbitrary number of objects to be stored.
- Class libraries usually contain tried-and-tested collection classes.
- Java's class libraries are called *packages*.
- We have used the `ArrayList` class from the `java.util` package.

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 3 37

Review

- Items may be added and removed.
- Each item has an index.
- Index values may change if items are removed (or further items added).
- The main `ArrayList` methods are `add`, `get`, `remove` and `size`.
- `ArrayList` is a parameterized or generic type.

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 3 38