Теми:

- **Взаимодействия на обекти.Понятие за абстракия и модулност.**
- **Програмни практики: цифров часовник: диаграми на класове и обекти; проект и програмни елементи.**
- **Групиране на обекти. Колекции и итератори. Програмни практики: проект ‘notebook’; обектна структура, използване на колекции.**

Object interaction

Creating cooperating objects

Abstraction and modularization

Definition:

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

Divide-and-conquer

An Object Distinction

- procedural model
- abstract data type (ADT) model
- object-oriented (OO) model

Procedural Model

An example of this is string manipulation using character arrays and the family of str* functions defined in the Standard C library:

```
char boy[] = "Danny";
char *p_son;

...

p_son = new char[ strlen( boy ) + 1 ];
strcpy( p_son, boy );

...

if ( !strcmp( p_son, boy ))
   take_to_disneyland( boy );
```

Abstract Data Type (ADT)

Users of the abstraction are provided with a set of operations (the public interface), while the implementation remains hidden. An example of this is a String class:

```
String girl = "Anna";
String daughter;

...

// String::operator=();
daughter = girl;

...

// String::operator==();
if ( girl == daughter )
   take_to_disneyland( girl );
```

Object-Oriented (OO) model

- A collection of related types are encapsulated through an abstract base class providing a common interface. An example of this is a Library_materials class from which actual subtypes such as Book, Video, Compact_Disc, Puppet, and Laptop are derived:

```
void
check_in( Library_materials *pmat )
{
    if ( pmat->late() )
        pmat->fine();
    pmat->check_in();
    if ( Lender *plend = pmat->reserved() )
        pmat->notify( plend );
}
```

Designing classes

Step 1: Design the attributes and behaviors of the object.

Step 2: Define the object. Create a class definition or a blueprint.

Step 3: Create an object.

Step 4: Use the object.

A construction company would like to handle a customer's order for a new home. The customer may select one of four models of a home. Model A for 10000, Model B for 120000, Model C for 180000, or Model D for 250000. Each model can have one, two, three, or four bedrooms.

Several nouns have been underlined in this description. Identify the possible objects that may be defined by this narrative. Not all the nouns underlined make sense when used as a potential object.

Modularizing the clock display

One four-digit display?

Or two two-digit displays?

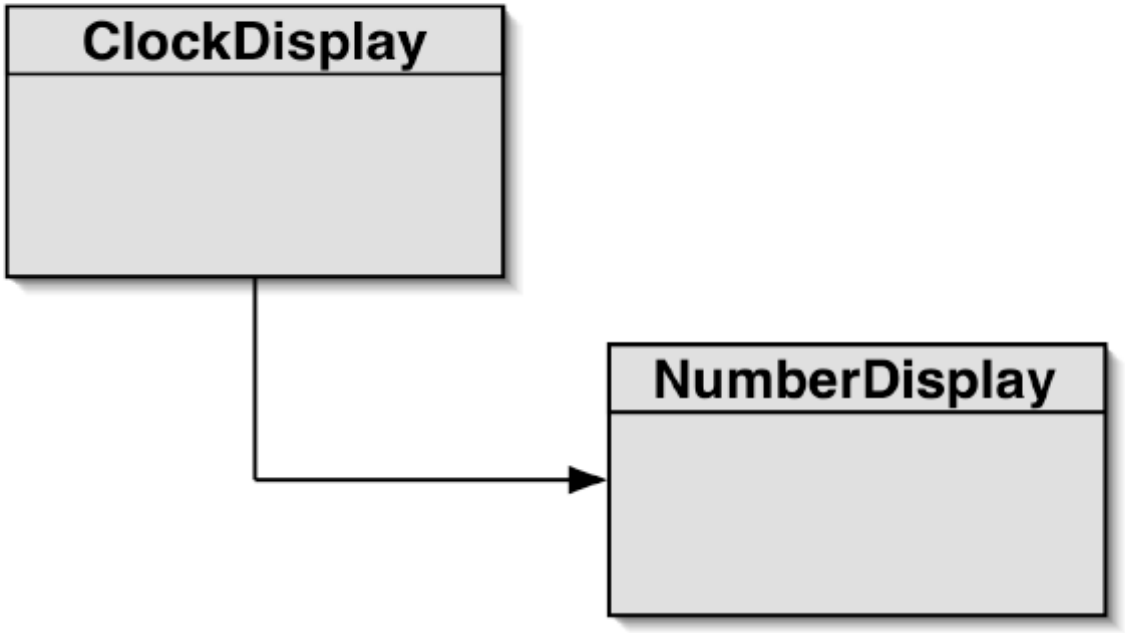
Implementation - NumberDisplay

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

    Constructor and
    methods omitted.
}
```

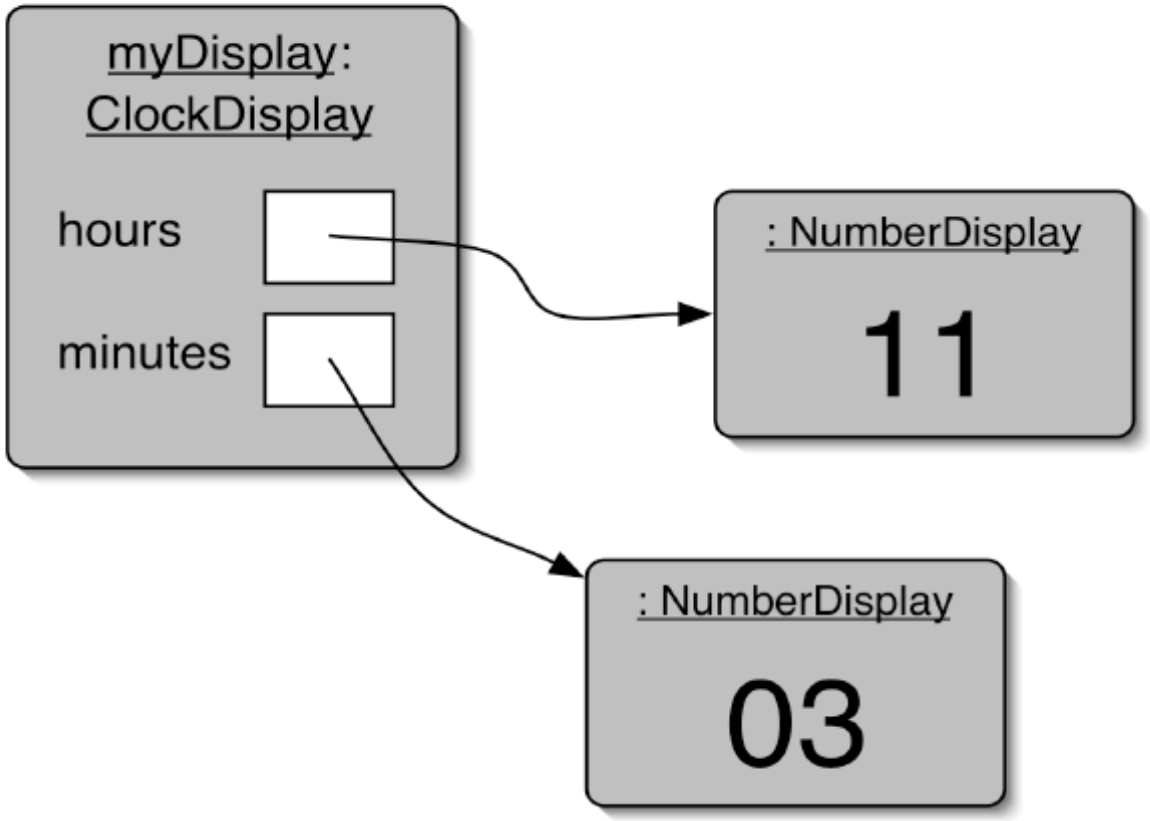
Implementation -ClockDisplay

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

    Constructor and
    methods omitted.
}
```

Class diagram



Object diagram



Fragments of

Source code: NumberDisplay

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

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

Source code: NumberDisplay

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

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();
    }
}
```

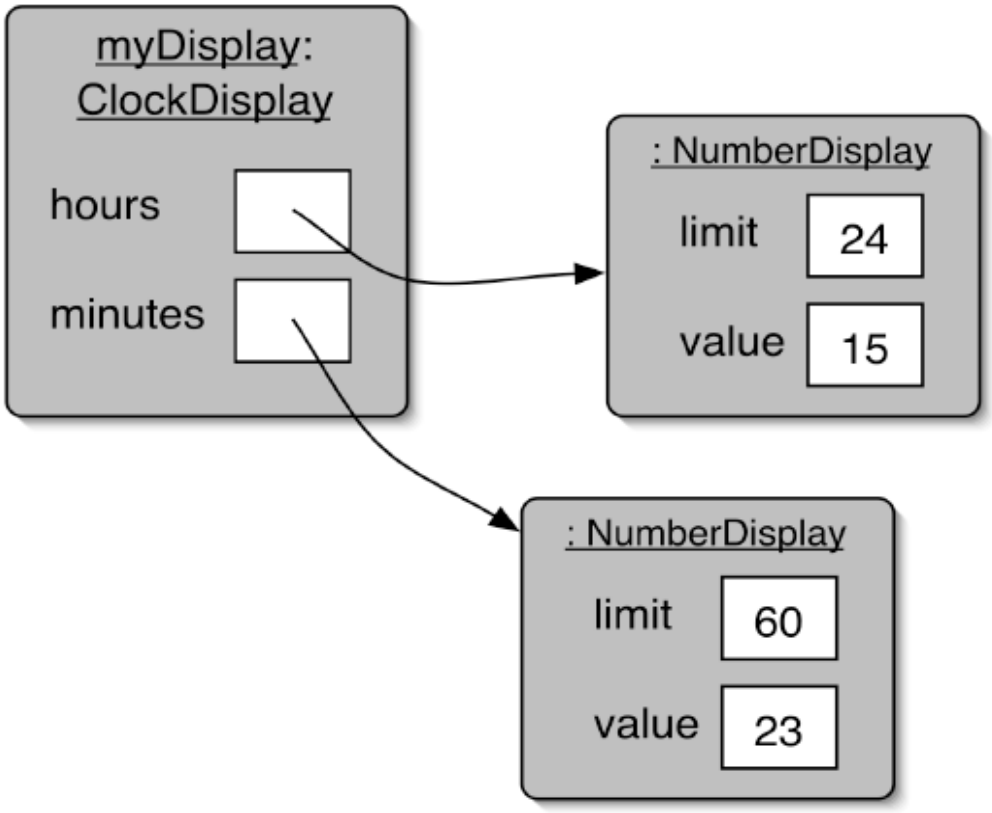
Method calling

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

Internal method

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

ClockDisplay object diagram



Objects creating objects

in class NumberDisplay:

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

in class ClockDisplay:

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

Method calls

- internal method calls

```
updateDisplay();
…
private void updateDisplay()
```

- external method calls

```
minutes.increment();
```

New theme

Grouping objects

Collections and iterators

Колекции от данни

- Колекциите са обекти, които сочат към група от обекти.
- Колекциите съдържат референции към данни от тип обект.
- Всякакъв тип обект може да се съхранява в колекция.

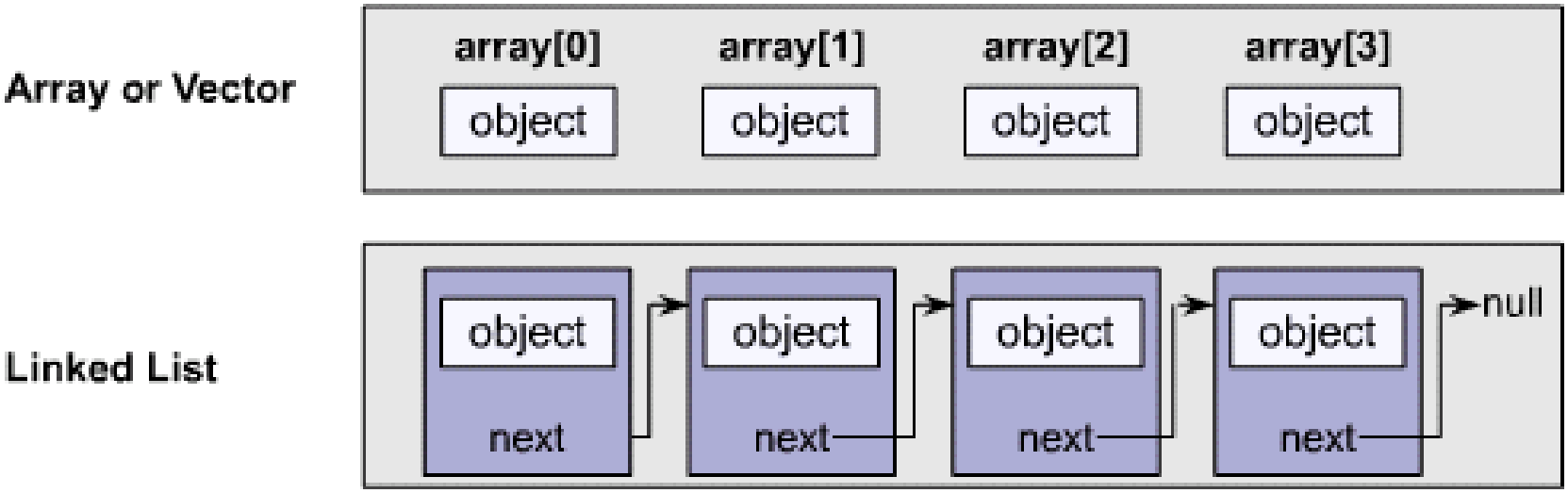
Collections Framework

- интерфейси
- имплементация
- алгоритми

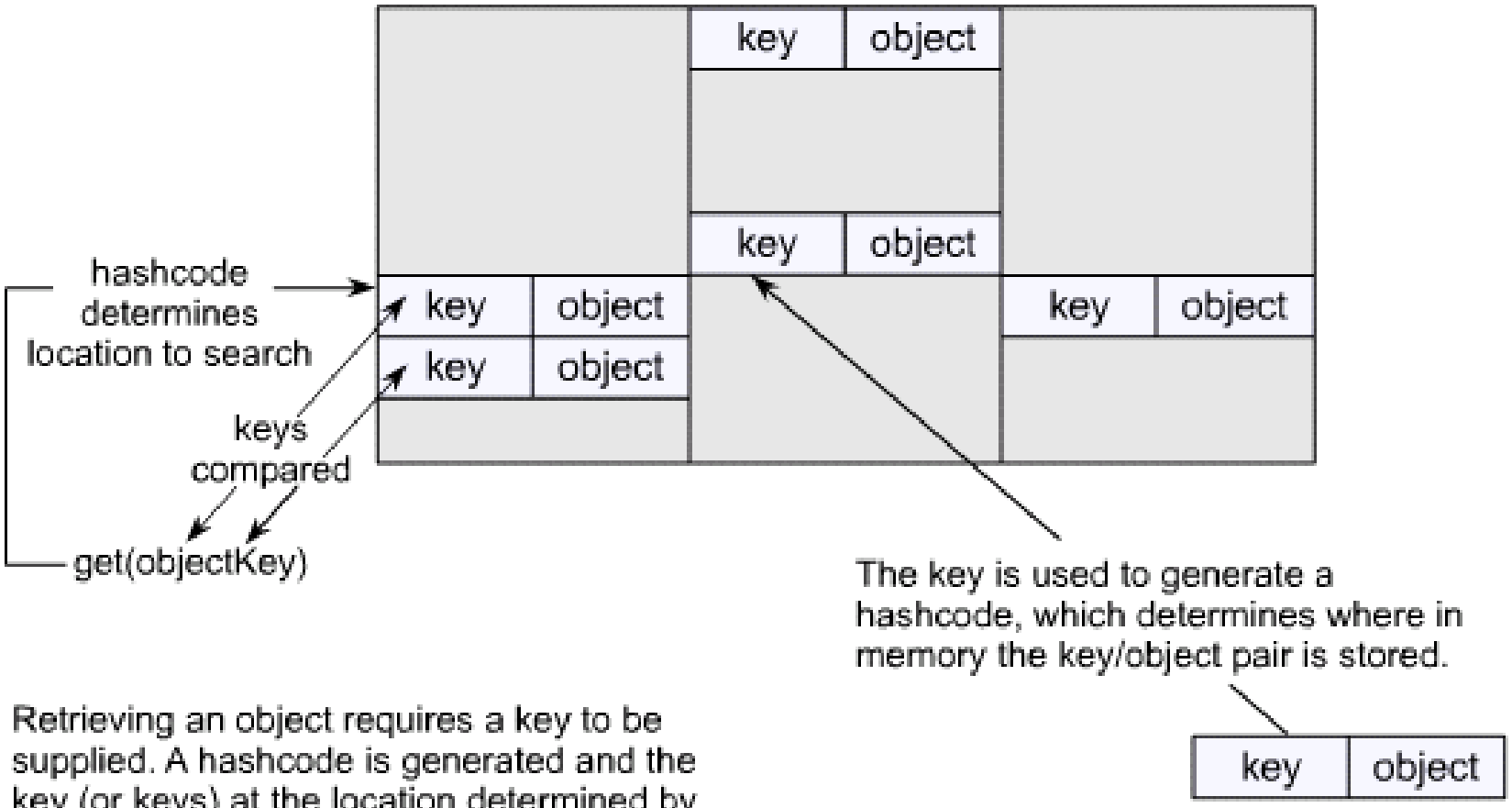
Технология на съхраняване на колекцията

- Масив (Array):
  - Фиксиран размер, бързо и ефикасно за достъп, трудно за модифициране.
- Свързан списък (Linked List):
  - Елементите имат референция за упътване към следващия елемент, лесен за промяна, бавен за претърсване.
- Дърво (Tree):
  - Лесен за промяна, съхранява елементите в ред.
- Хеш таблица (Hashtable):
  - Използва се за индексиране на ключ който идентифицира елементите. Елементите са извличат от хеш таблицата чрез използване на ключ на елемента.

Масиви и свързан списък



Хеш таблица



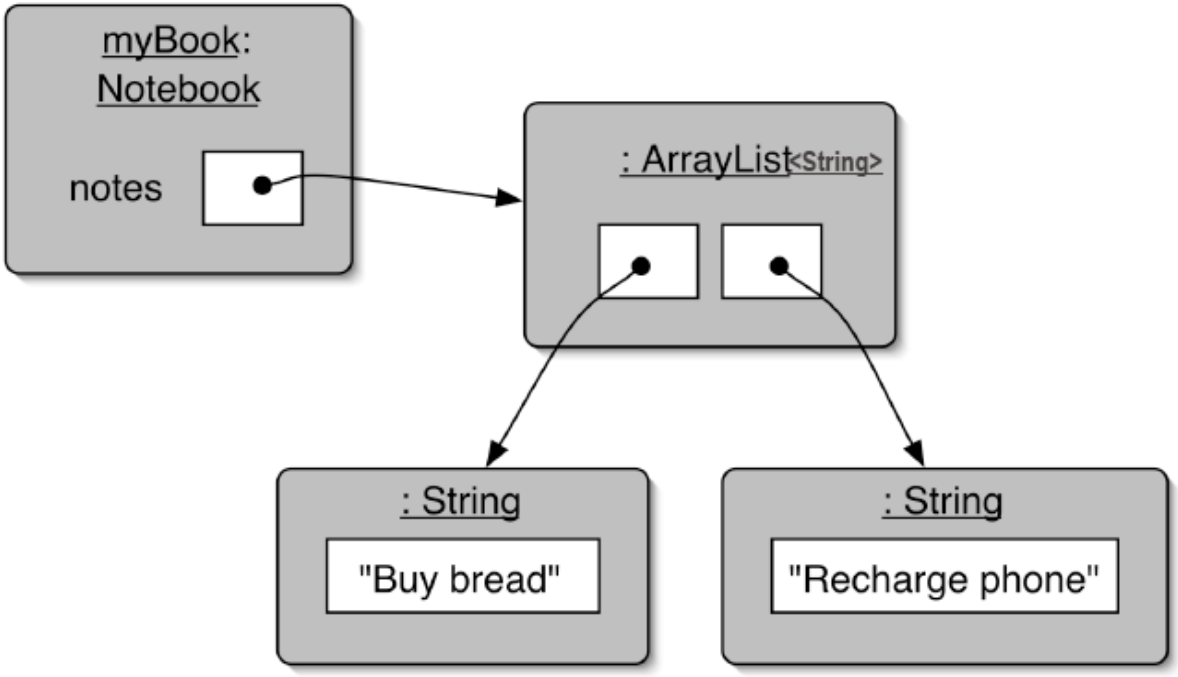
Retrieving an object requires a key to be supplied. A hashcode is generated and the key (or keys) at the location determined by the hashcode is compared with the supplied key.

Видове колекции

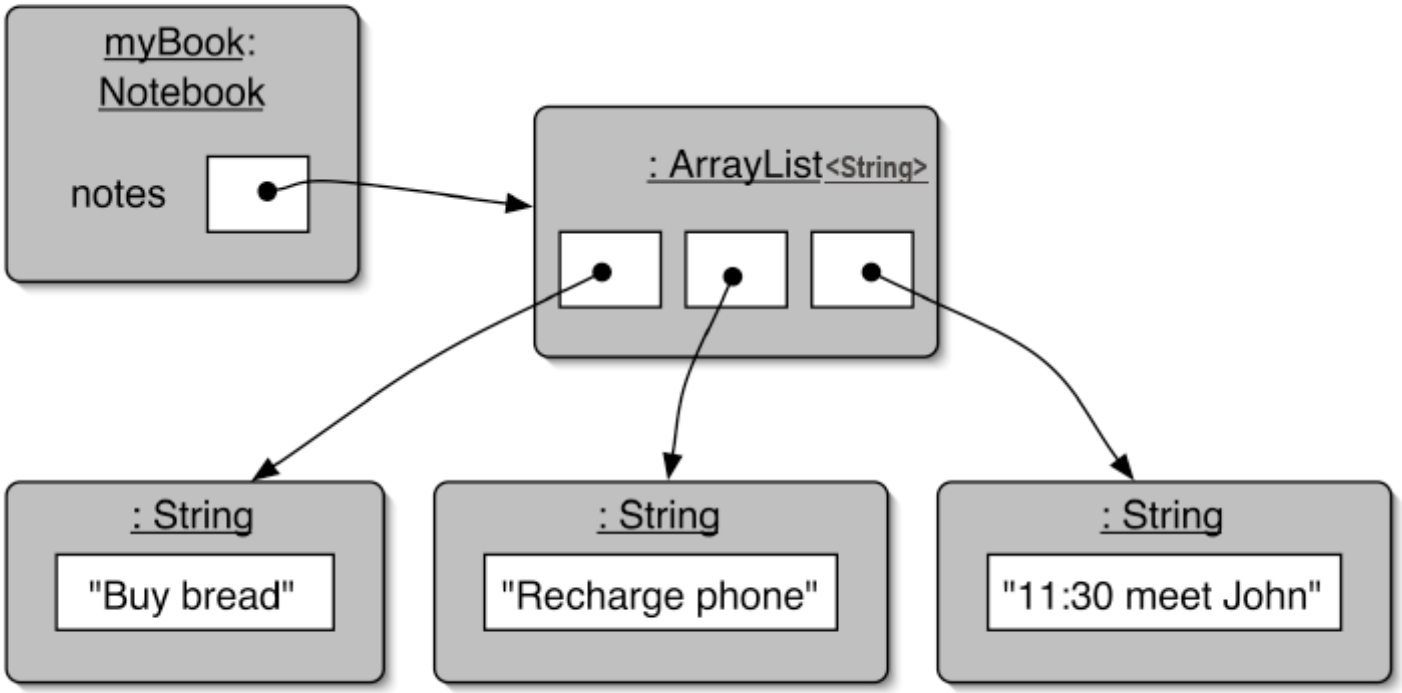
- Collection
  - Прост контейнер от неподредени обекти. Повторенията са позволени.
- List
  - Контейнер от подредени елементи. Повторенията са позволени.
- Set
  - Неподредена колекция от обекти, в която повторенията не са позволени.
- Map
  - Колекция от ключ/стойност по двойки. Ключът се използва за индекс на елемента. Повторение на ключове не се допуска.

```
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 structures with collections



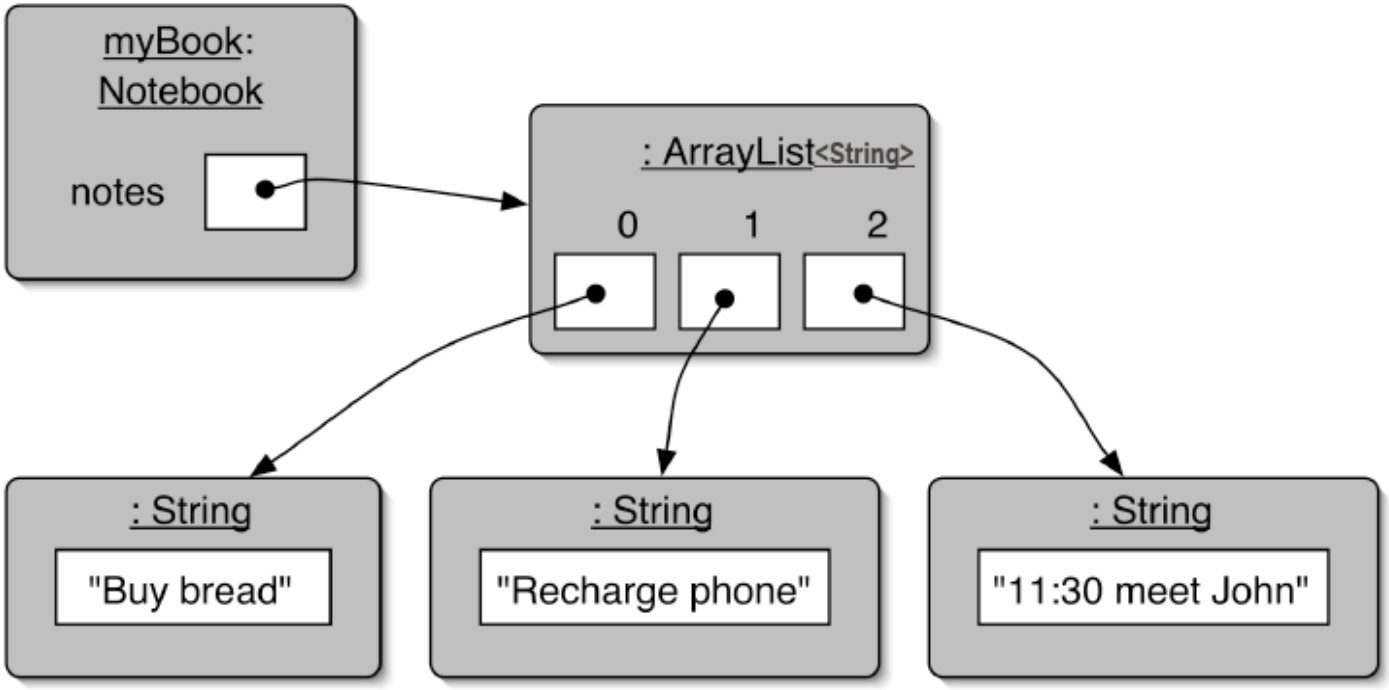
Adding a third note



Using the collection

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

    public void storeNote(String note)
    {
        notes.add(note);
    }

    public int numberOfNotes()
    {
        return notes.size();
    }

    ...
}
```

Index numbering



Retrieving an object

```
public void showNote(int noteNumber)
{
    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.
    }
}
```