

Introduction to Programming

with Java, for Beginners

List

Collections

- Data: Groups of entities of similar kind
 - CD collection
 - Items in store
 - Characters in a video game
- Methods: Do the “same thing” to all of them
 - Computing the total number of tracks
 - Adding sales tax to all the prices
 - Moving all of them one step forward

2

Array Recap

- Kind of a collection


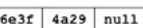

```
Person[] people;
```

people 

```
people = new Person[3];
```

people  → 

```
people[0]=new Person("jo");  
people[1]=new Person("flo");
```

people  → 


3

Problems with Arrays

- Fixed size
 - Optimal size may depend on information unavailable at creation time
 - Not good with dynamic growth
 - Cannot add, and remove elements without the pain of resizing
- A collection that can dynamically resize is known as a *list*

4

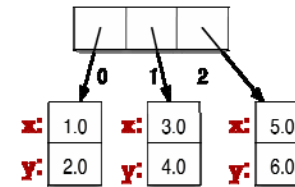
Lists

- *Basic requirements*
 - Add an element
 - Get an element
 - Remove an element
 - Search for an element
 - Learn the size of the list
 - Perform an operation on all elements
- We will look at:
 - How to create a type-specific List (for Points, Strings)
 - A much more convenient List, the one provided in the API, that works on *any* type

5

Visualizing a list

- A sequence of references to objects of a given type
- Indexed by 0, 1, 2, ...
- A list of **Points**:



6

A Type-Specific List (for Points)

```
public boolean add(Point p)
    Add p to the list. Returns false if duplicates are not permitted and already contains
    the specified element

public boolean remove(Point p)
    Remove the first occurrence of p. Returns true if this collection contained the
    specified element

public boolean contains(Point p)
    Test whether the list contains p

public int size()
    Number of elements in list

public Point get(int i)
    Get the ith element of the list (starts from 0)
    0 <= i && i < size()

public void set(int i, Point p)
    Set the ith element to p
    0 <= i && i < size()
```

7

Using a type-specific List

```
> PointList points = new PointList();
> points.add(new Point(1.0,2.0));
> points.add(new Point(3.0,4.0));
> points.add(new Point(5.0,6.0));
> points.size()
3
> points.get(2).getX()
5.0
```

8

Now a list of Strings...

```
public boolean add(String s)
    Add s to the list
public boolean remove(String s)
    Remove the first occurrence of s
public boolean contains(String s)
    Test whether the list contains s
public int size()
    Number of elements in list
public String get(int i)
    Get the ith element of the list (starts from 0)
    0 <= i && i < size()
public void set(int i, String s)
    Set the ith element to s
    0 <= i && i < size()
```

9

Problems: Code Duplication; Lack of abstraction

- Much of the functionality of a list doesn't depend on the element type. E.g.: `size`, `add`, `remove`, `swap`, etc.

```
public void swap(PointList list, int i, int j) {
    Point p = list.get(i);
    list.set(i, list.get(j));
    list.set(j, p);
}

public void swap(StringList list, int i, int j) {
    String s = list.get(i);
    list.set(i, list.get(j));
    list.set(j, s);
}
```

10

The java.util.List interface

- The Java API's List interface works with any Object

```
public interface List {
    public int size();
    public boolean isEmpty();
    public boolean add(Object element);
    public void add(int i, Object element);
    public Object get(int i);
    public Object remove(int i);
    public boolean contains(Object e);
    public Object set(int i, Object element);
    . . .
}
```

Cannot do: `List myList = new List()`

11

ArrayList

- `ArrayList` is a concrete classes
- Implements the `List` interface
- Does dynamic resizing for you
- Typical Usage:
 - Use `List` for variables and parameters
 - Use `ArrayList` when instantiating the list
`List cdTracks = new ArrayList();`

12

Typical Usage of *List* interface

- Example: ArrayList with Strings

```
> import java.util.*;
> List words = new ArrayList();
> words.add("cat");
> words.add("rat");
> words.get(1)
"rat"
```

- One approach for processing all items in a List:

```
for (int i = 0; i < words.size(); i++)
    System.out.println(((Object) words).get(i));
```

13

Some more problems

- Use of the type Object as the basic type for objects that are stored in a list leads
 - Need to use type-casting in almost every case when an element is retrieved from that list
 - Since any type of object can legally be added to the list, there is no way for the compiler to detect an attempt to add the wrong type of object to the list
 - The error will be detected only at run time i.e. when the object is retrieved from the list and the attempt to type-cast the object fails

14

Solution: Parameterized Types

- An array of type BaseType[] can **only** hold objects of type BaseType
 - An attempt to store an object of the wrong type in the array will be detected by the compiler
 - There is no need to type-cast items that are retrieved from the array back to type BaseType.
- Instead of using the plain "List" type, it is possible to use **List<BaseType>**
 - Where BaseType is any object type, that is, the name of a class or of an interface
 - BaseType **cannot** be one of the primitive types
 - Used to create lists that can hold only objects of type BaseType. For example,
 - List<Point> p1 = new ArrayList<Point>();
 - ArrayList<Point> p2 = new ArrayList<Point>();

15