

Grouping Objects

Collections

Often we want to deal with many objects of the same **type**:

- Books in a library
- Students in a course
- Houses on a street

We call these **collections**.

Collections

Java has built-in support for many kinds of collections:

- Variable-length ordered lists
- Fixed-length arrays
- Sets
- Dictionaries

The Java Class Library

Java comes with a **huge** library of useful classes.

We usually call this the **Java API**.

<http://download.oracle.com/javase/6/docs/api/>

(API stands for **Application Programming Interface** but most people forget that.)

ArrayList

In the java.util package there is a class called **ArrayList**.

This class is used to represent variable length lists of objects.

<http://download.oracle.com/javase/6/docs/api/java/util/ArrayList.html>

ArrayList Constructors

The ArrayList has three constructors:

```
// create an empty list
```

```
public ArrayList();
```

```
// empty list with reserve space
```

```
public ArrayList(int capacity);
```

```
// copy another collection
```

```
public ArrayList(  
    Collection<? extends E> c);
```

ArrayList methods

The ArrayList has many methods.

A few important ones:

```
public int size();
```

```
public boolean contains();
```

```
public E get(int index);
```

```
public E set(int index, E entry);
```

```
public boolean add(E entry);
```

```
public E remove(int index);
```

Generic classes

Each container holds a specific **type** (or class) of object.

This class is given as a **type parameter** in angle brackets `<>` after the name.

The type of the methods **get**, **set** and **add** depend on the given type.

Parameterised types are called **generic classes**.

List of Books

```
ArrayList<Book> library =  
    new ArrayList<Book>();  
  
Book b1 = new Book("Godot");  
  
library.add(b1);  
  
Book b2 = library.get(0);
```

List of ints

This **does not work**:

```
ArrayList<int> scores =  
    new ArrayList<int>();  
  
scores.add(1);  
  
int s = scores.get(0);
```

The type parameter must be a **class**
not a **primitive type**.

List of ints

Instead we use:

```
ArrayList<Integer> scores =  
    new ArrayList<Integer>();  
  
scores.add(1);  
  
int s = scores.get(0);
```

The Integer class is an **object-wrapper** for the primitive type **int**.

Count like a programmer

When accessing objects from a list, we start counting at **zero** not **one**.

```
list.get(0);  
// returns 35
```

```
list.get(1);  
// returns 2
```

```
list.size(); // returns 4
```

```
list.get(4); // ERROR
```

0	1	2	3
35	2	-3	100

Loops

Often in our code we want to perform a sequences of statement **multiple times**.

```
System.out.println(library.get(0));  
System.out.println(library.get(1));  
System.out.println(library.get(2));  
System.out.println(library.get(3));
```

Loop statements allow us to **abbreviate** this to a single statement.

Loops

There are several different kinds of loops:

- **for-each loops** - do an operation on each item in a collection
- **while loops** - do an operation until a condition changes
- **for-loops** - do an operation a specified number of times

for-each loop

The **for-each loop** is used to iterate over all the elements of a collection:

```
for (Class item : collection) {  
    // do some operation on item  
}
```

while loop

Iterators

Removing objects