

Object references

Arrays

Recall that an **int** array is declared in the following way:

```
int[] numbers = new int[size];
```

This is actually an abbreviation of

```
int[] numbers;  
numbers = new int[size];
```

Something like this actually happens when you instantiate an object.

Object instantiation

To instantiate an object,

```
Dog lucy = new Dog();
```

This is actually an abbreviation of

```
Dog lucy;  
lucy = new Dog();
```

Reference variables

- `Dog lucy;`—This creates a *reference variable* for a `Dog` object. Its purpose is to store the address of a `Dog` object.
- `lucy = new Dog();`—This instantiates an actual `Dog` object and its address is stored in `lucy`.

In particular, `lucy` itself is not a `Dog` object. It is a *reference* that points to a `Dog` object.

Parameters and arguments

Many methods have parameters. For example, consider:

```
public void moo(int a, int b) {  
    ...  
}
```

When calling this method, a caller must pass two **int** values for parameters *a* and *b*.

Parameters and arguments

A variable inside a method whose value is to be specified by a caller is called a *parameter*.

A value being passed by a caller to a parameter is called an *argument*.

A parameter is always a *variable*.

An argument is always a *value*.

Parameters and arguments

Consider the class `Coo` and the method:

```
public void moo(Coo a, Coo b) {  
    ...  
}
```

When calling this method with

```
c.moo(x, y);
```

the value of `x` and `y` are passed, where these values are addresses of `Coo` objects.

Variables

Example. What does this output?

```
int x = 0;
```

```
int y = 1;
```

```
x = y;
```

```
x = 2;
```

```
System.out.print(x);
```

```
System.out.print(y);
```

Output: 2 1

Object references

Example. What does this output?

```
Dog lucy;  
Dog sasha;  
lucy = new Dog("Golden Retriever", ...);  
sasha = lucy;  
sasha.setBreed("German Sephard");  
System.out.print(lucy.getBreed());  
System.out.print(sasha.getBreed());
```

Output: German Sephard German Sephard

Data structures

Data structures

What is a data structure?

Simple answer: a collection of data equipped with some operations.

Examples

- Lists
- Strings
- ...

Data structures

Data structures are formalized in *abstract data types* (ADTs) using the *object-oriented paradigm*.

- An ADT is abstractions of data and basic operations on the abstractions.
- We will use a Java class to implement an ADT.

Lists

Lists

- Lists store arbitrary objects in linear structures.
- Arbitrary element access, insertion and removal are permitted.
- We will consider two distinct ways to access elements:

Index — an integer-valued location.

Link — a relative reference.

The ADT list

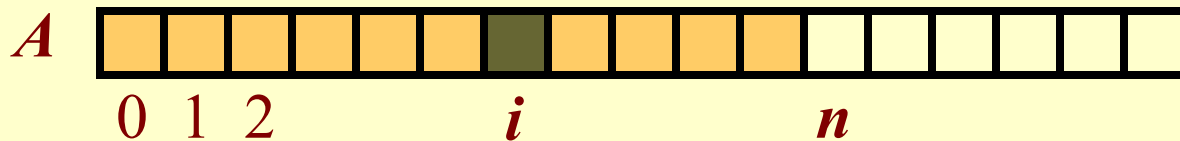
- The ADT *list* extends the notion of arrays.
- An element can be accessed, inserted or removed by specifying its *index*: the number of elements preceding it.
- Elements are stored in contiguous cells but not necessarily sorted.

Main list operations

- ***get(i)***: returns the element at index i without removing it.
- ***set(i, e)***: replaces the element at index i with a new element e and returns the old element.
- ***add(i, e)***: inserts a new element e at index i .
- ***remove(i)***: removes and returns the element at index i .

Array-based implementation

- Naturally, we use an array, say A .
- An element of index i is stored in $A[i]$.
- We also need an integer variable to keep track of the number of elements stored.



ArrayList

```
public class ArrayList<E> {  
  
    public static final int CAPACITY = 16;  
    private E[] data;  
    private int size = 0;  
  
    public ArrayList() { this(CAPACITY); }  
    public ArrayList(int capacity) {  
        data = (E[]) new Object[capacity];  
    }  
  
    ...  
}
```

ArrayList

```
public class ArrayList<E> {  
  
    ...  
  
    public E get(int i) { return data[i]; }  
  
    public E set(int i, E e) {  
        E toReturn = data[i];  
        data[i] = e;  
        return toReturn;  
    }  
  
    ...  
  
}
```

ArrayList

```
public class ArrayList<E> {  
  
    ...  
  
    public void add(int i, E e) {  
        if (size == data.length)  
            resize(2*data.length);  
        for (int k = size - 1; k >= i; k--)  
            data[k + 1] = data[k];  
        data[i] = e;  
        size++;  
    }  
  
    ...  
  
}
```

ArrayList

```
public class ArrayList<E> {  
  
    ...  
  
    public E remove(int i) {  
        E toReturn = data[i];  
        for (int k = i; k < size - 1; k++)  
            data[k] = data[k + 1];  
        size--;  
        return toReturn;  
    }  
  
    ...  
  
}
```

ArrayList

```
public class ArrayList<E> {  
  
    ...  
  
    private void resize(int capacity) {  
        E[] A = (E[]) new Object[capacity];  
        for (int k = 0; k < size; k++)  
            A[k] = data[k];  
        data = A;  
    }  
  
    ...  
  
}
```

ArrayList

An array list is declared as an object in the following way for a class `E`:

```
ArrayList<E> list = new ArrayList<E>();
```

This is actually an abbreviation of

```
ArrayList<E> list;  
list = new ArrayList<E>();
```

ArrayList

Example. To declare an array list of integers named `numbers`, we put, using Java's class `Integer` for integers,

```
ArrayList<Integer> numbers = new ArrayList<Integer>();
```

This is an abbreviation of

```
ArrayList<Integer> numbers;  
numbers = new ArrayList<Integer>();
```

ArrayList

Example. To declare an array list of dogs named `dogs`, we put, using the class `Dog` for dogs,

```
ArrayList<Dog> dogs = new ArrayList<Dog>();
```

This is an abbreviation of

```
ArrayList<Dog> dogs;  
dogs = new ArrayList<Dog>();
```

Printing entries

Example. A **for** loop to print all integers in the array list `numbers`:

```
for (int i = 0; i < numbers.getSize(); i++)  
    System.out.println(numbers.get(i));
```

Printing entries

Example. A **for** loop to print all dogs' breeds in the array list `dogs`:

```
for (int i = 0; i < dogs.getSize(); i++)  
    System.out.println(dogs.get(i).getBreed());
```

Printing entries

Example. A **for** loop to print all dogs' ages in the array list `dogs`:

```
for (int i = 0; i < dogs.getSize(); i++)  
    System.out.println(dogs.get(i).getAge());
```

Searching

Example. A **for** loop to find the *maximum* integer in the array list `numbers`:

```
int maxSoFar = MININT;
for (int i = 0; i < numbers.getSize(); i++)
    if (numbers.get(i) > maxSoFar)
        maxSoFar = numbers.get(i);
System.out.println("Max: " + maxSoFar);
```

Searching

Example. A **for** loop to find the *oldest age* of a dog in the array list `dogs`:

```
int oldestSoFar = 0;
for (int i = 0; i < dogs.getSize(); i++)
    if (dogs.get(i).getAge() > oldestSoFar)
        oldestSoFar = dogs.get(i).getAge();
System.out.println("Oldest age: " + oldestSoFar);
```

Searching

Example. A **for** loop to find the *heaviest* weight of a dog in the array list `dogs`:

```
int heaviestSoFar = 0;
for (int i = 0; i < dogs.getSize(); i++)
    if (dogs.get(i).getWeight() > heaviestSoFar)
        heaviestSoFar = numbers.get(i).getWeight();
System.out.println("Heaviest weight: " + heaviestSoFar);
```