Programming in the Real World

Ceçi n'est pas une Java

```
import acm.program.*;

public class MyProgram extends ConsoleProgram {
    public void run() {
        println("Hello, world!");
    }
}
```

The ACM Libraries

- Throughout this class we've been using the ACM libraries.
  - `acm.program.*`
    - `ConsoleProgram`, `GraphicsProgram`, etc.
  - `acm.graphics.*`
    - `GOval`, `GRect`, etc.
  - `acm.util.*`
    - `RandomGenerator`
    - `ErrorException`

The ACM Libraries

- The ACM libraries exist to simplify many common Java techniques.
- However, the ACM libraries aren't widely used outside of CS106A.
- Good news: The topics from the latter half of the quarter (file reading, arrays, `ArrayList`, `HashMap`, interactors, etc.) use only standard Java.
- We do need to cover a few last-minute details of the Java language.

“Hello, World” Without the ACM

Starting up the Program

- In standard Java, program execution begins inside a method called

```
public static void main(String[] args)
```

- The ACM libraries contain this method in the `Program` class.
- When you're not using the ACM libraries, you will have to implement this method yourself.

Starting up the Program

In standard Java, program execution begins inside a method called

```
public static void main(String[] args)
```

The ACM libraries contain this method in the `Program` class.

When you're not using the ACM libraries, you will have to implement this method yourself.

What About Windows?

Steps to Create a Window

- Create a new `JFrame`, which actually represents the window object.
- Add any components or interactors to the frame as you normally would.
- Set the size of the window by calling

```
frame.setSize(width, height)
```

- Tell Java to quit when we close the program by calling

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)
```

- Show the window by calling

```
frame.setVisible(true)
```

What about Graphics?

- You can create components that can display graphics by extending `JComponent` and writing

```
public void paintComponent(Graphics g)
```

- You can then call methods to draw on the window when the window is resized or moved.
- Note: the default graphics system is not object-oriented.

`static` Methods

- A ***static method*** is a method that's specific to a *class*, rather than *instances* of that class.
- Examples:
  - `Character.isLetter`
  - `RandomGenerator.getInstance`
- Because the method is specific to the class rather than any instance, there is no receiver object.

`public static void main`

- Because `main` is **`static`**, there is no instance of your class that it operates relative to.
- Common technique: Have `main` create an instance of the class and work from there.
- This is done automatically by the ACM libraries.

How are you supposed to remember all these methods?

http://docs.oracle.com/javase/7/docs/api/

Time-Out for Announcements!

Assignment 7

- Assignment 7 (NameSurfer) due today at 3:15PM.
  - Due Wednesday at 3:15PM with one late period.
  - Due Friday at 3:15PM with two late periods.
  - Hard deadline: next Monday at 3:15PM.

A Note on Happiness and Staying Sane

Assignment 8

- Assignment 8 (**FacePamphlet**) goes out today and is due next Tuesday, March 17 at 8:30AM.
  - Put everything together and build a social network!
  - Only one new concept tested (iterators), and we'll talk about them in a second.
  - Our hope: This gives you a way to get extra practice with the material and increase your average assignment grade.
- Note the unusual due date.
  - ***This is a hard deadline – no late submissions will be accepted.***
  - You cannot use late periods on this assignment.

Iterators

- To visit every element of a collection, you can use the "for each" loop:

```
for (ElemType elem: collection) {
    …
}
```

- Alternatively, you can use an ***iterator***, an object whose job is to walk over the elements of a collection.
- The iterator has two commands:
  - `hasNext()`, which returns whether there are any more elements to visit, and
  - `next()`, which returns the next element and moves the iterator to the next position.

Java Iterators

```
ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    int curr = iter.next();

    /* … use curr … */
}
```

Java Iterators

```
ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    int curr = iter.next();

    /* … use curr … */
}
```

Java Iterators


```
ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    int curr = iter.next();

    /* … use curr … */
}
```

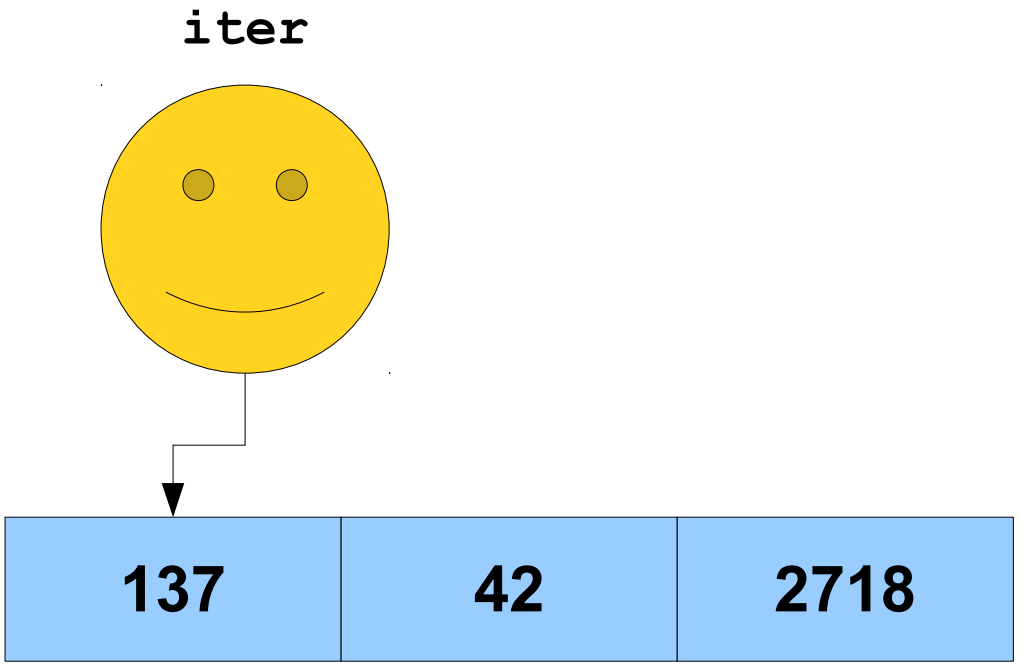
Java Iterators


```
ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    int curr = iter.next();

    /* … use curr … */
}
```

Java Iterators

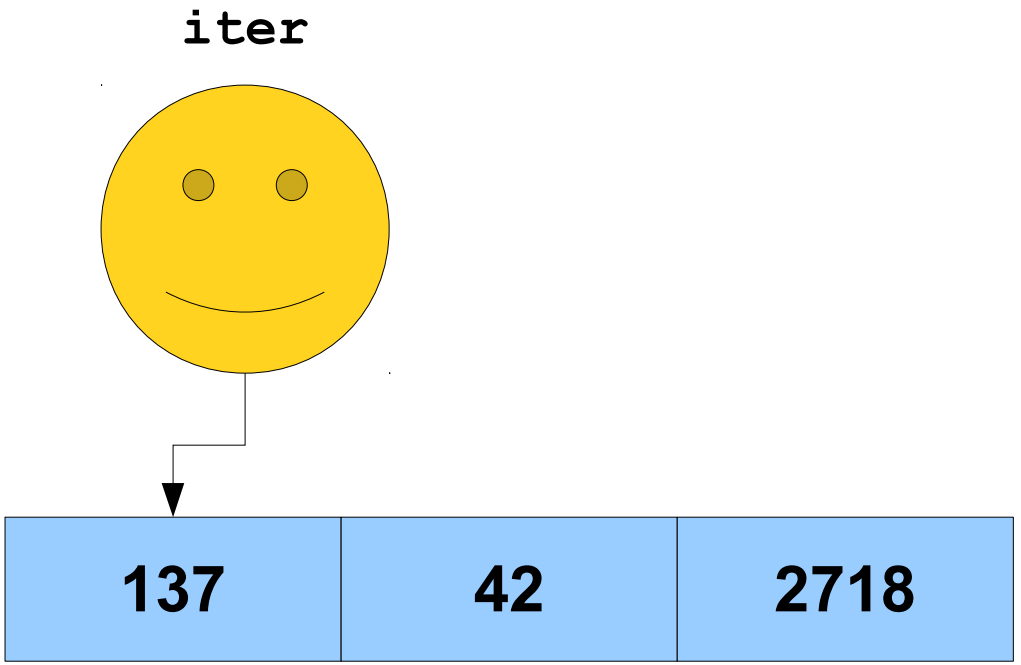



```
ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    int curr = iter.next();

    /* … use curr … */
}
```

Java Iterators

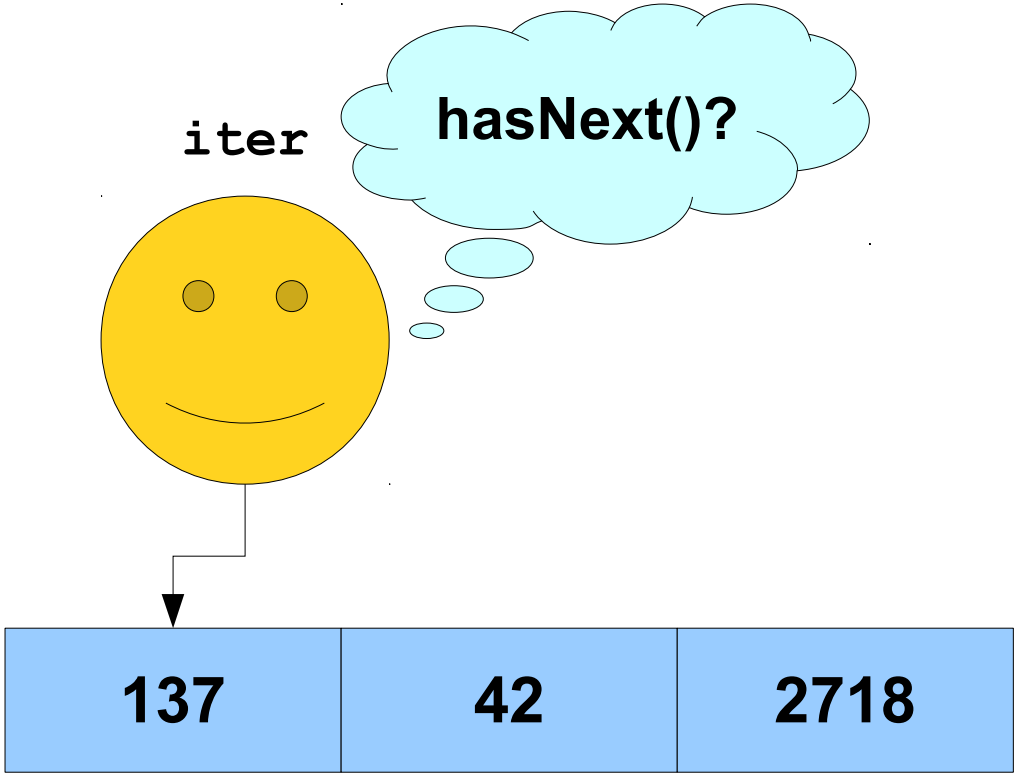



```
ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    int curr = iter.next();

    /* … use curr … */
}
```

Java Iterators

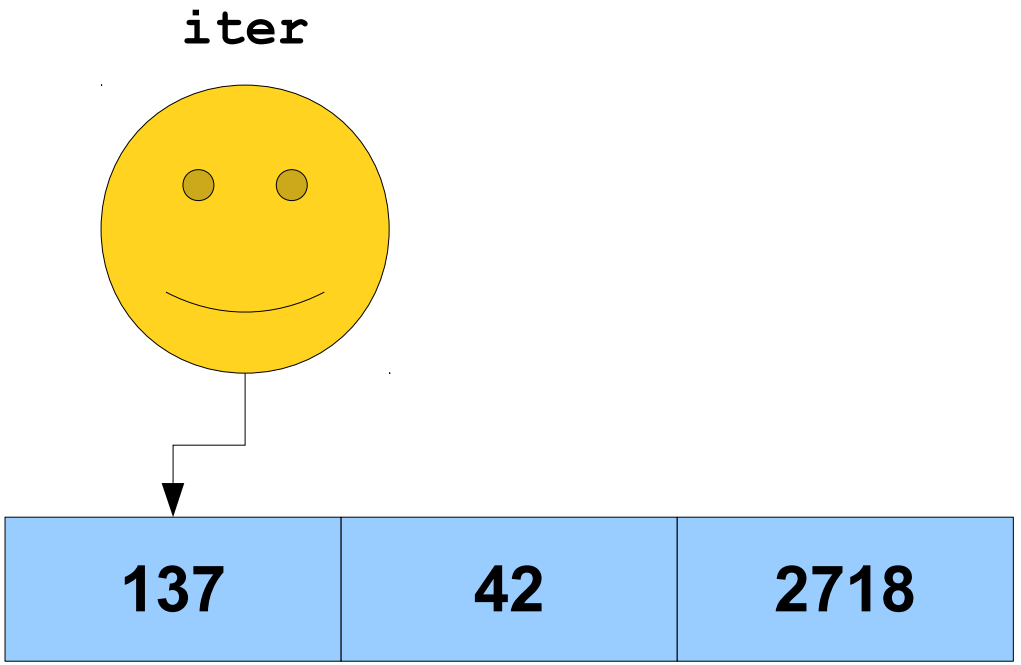



```
ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    int curr = iter.next();

    /* … use curr … */
}
```

Java Iterators

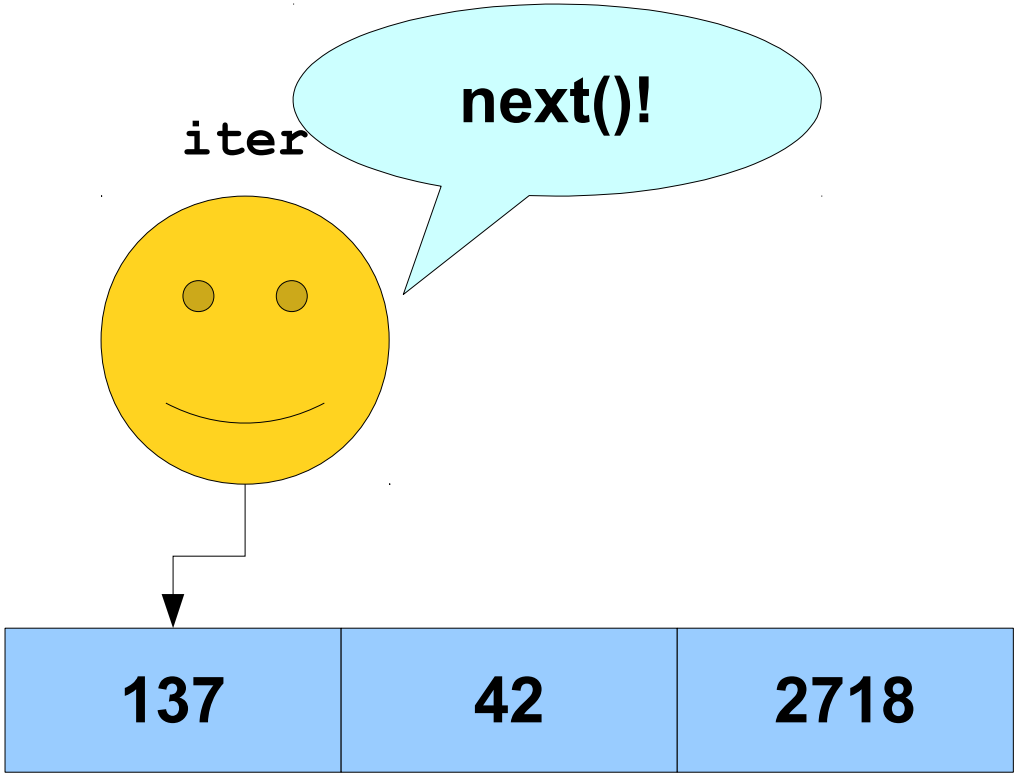



```
ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    int curr = iter.next();

    /* … use curr … */
}
```

Java Iterators

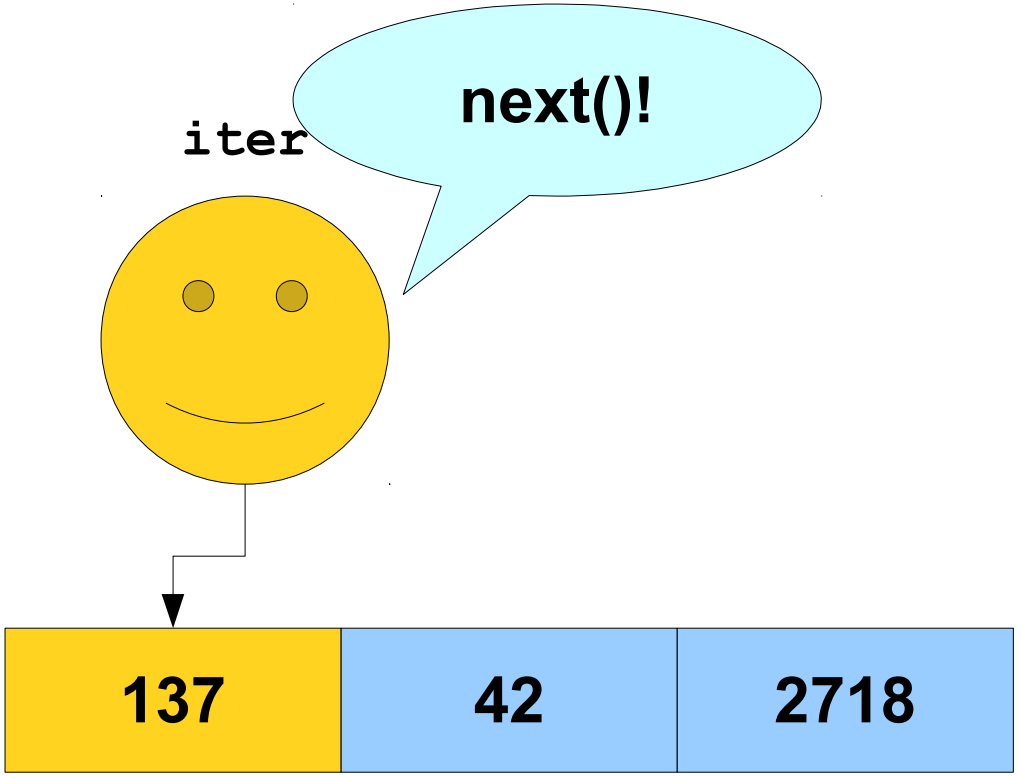



```
ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    int curr = iter.next();

    /* … use curr … */
}
```

Java Iterators

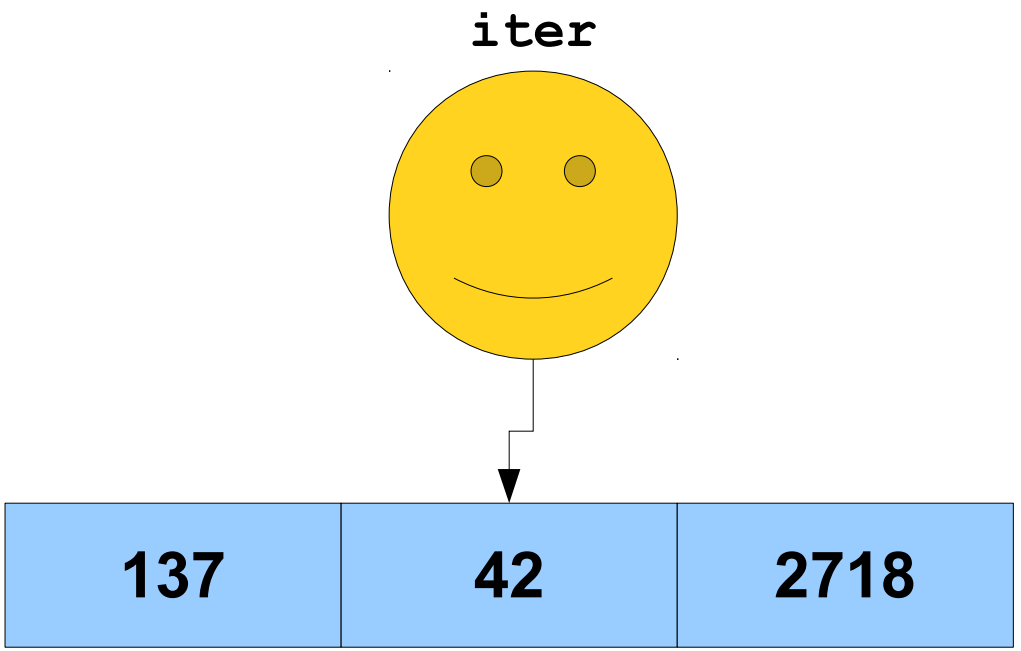



```
ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    int curr = iter.next();

    /* … use curr … */
}
```

Java Iterators

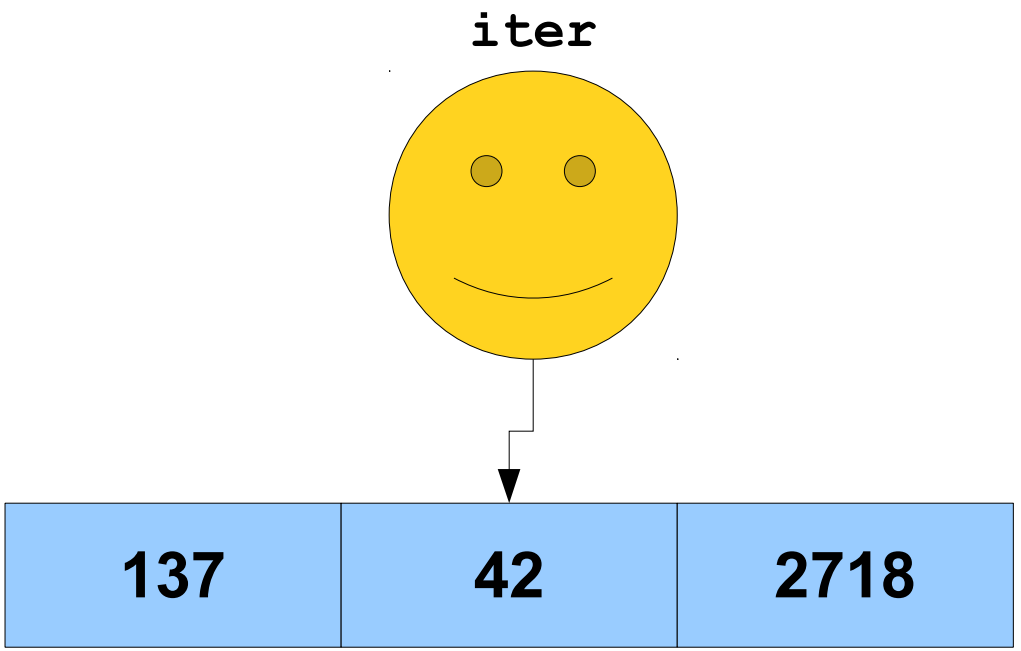



```
ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    int curr = iter.next();

    /* … use curr … */
}
```

Java Iterators

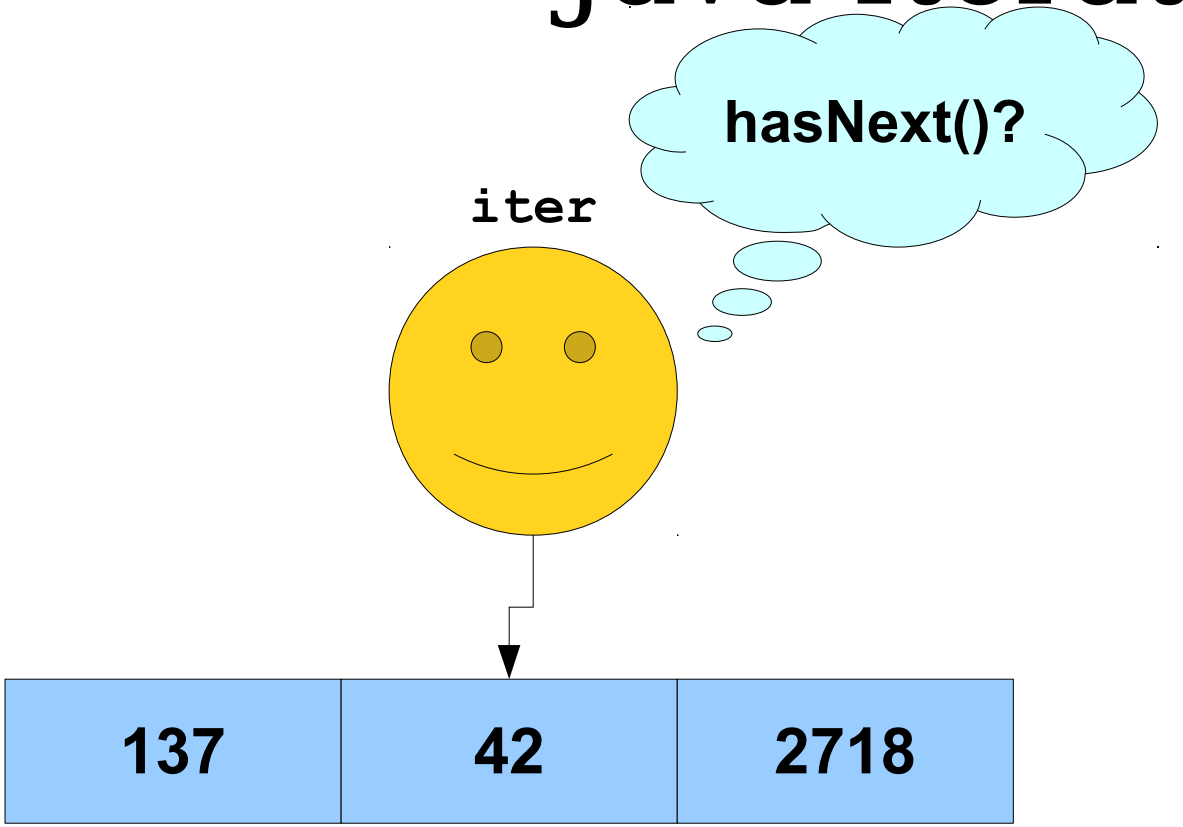



```
ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    int curr = iter.next();

    /* … use curr … */
}
```

Java Iterators

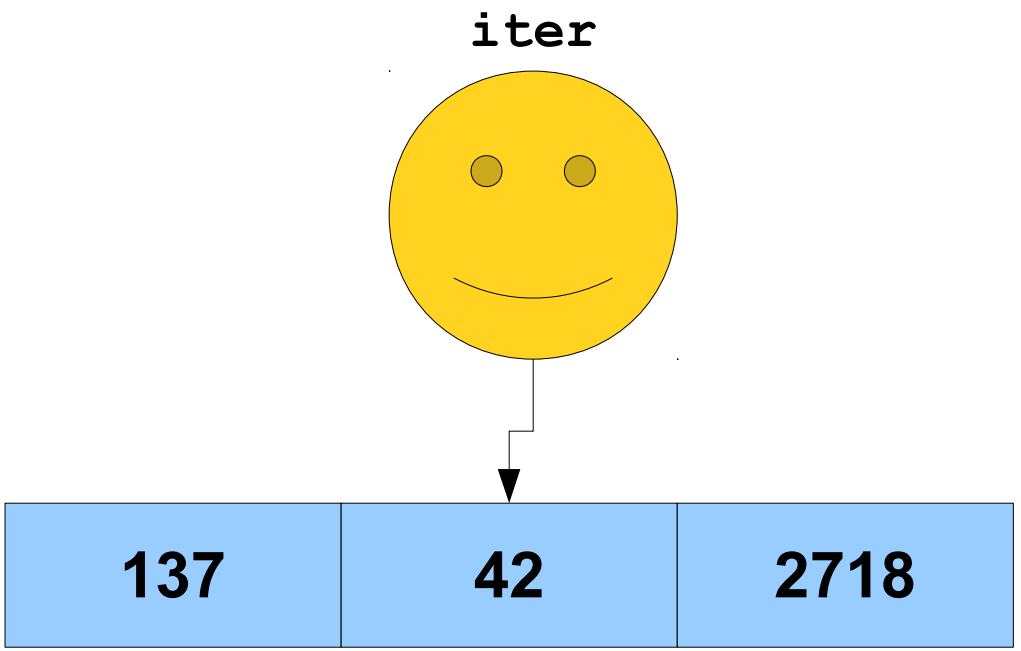



```
ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    int curr = iter.next();

    /* … use curr … */
}
```

Java Iterators

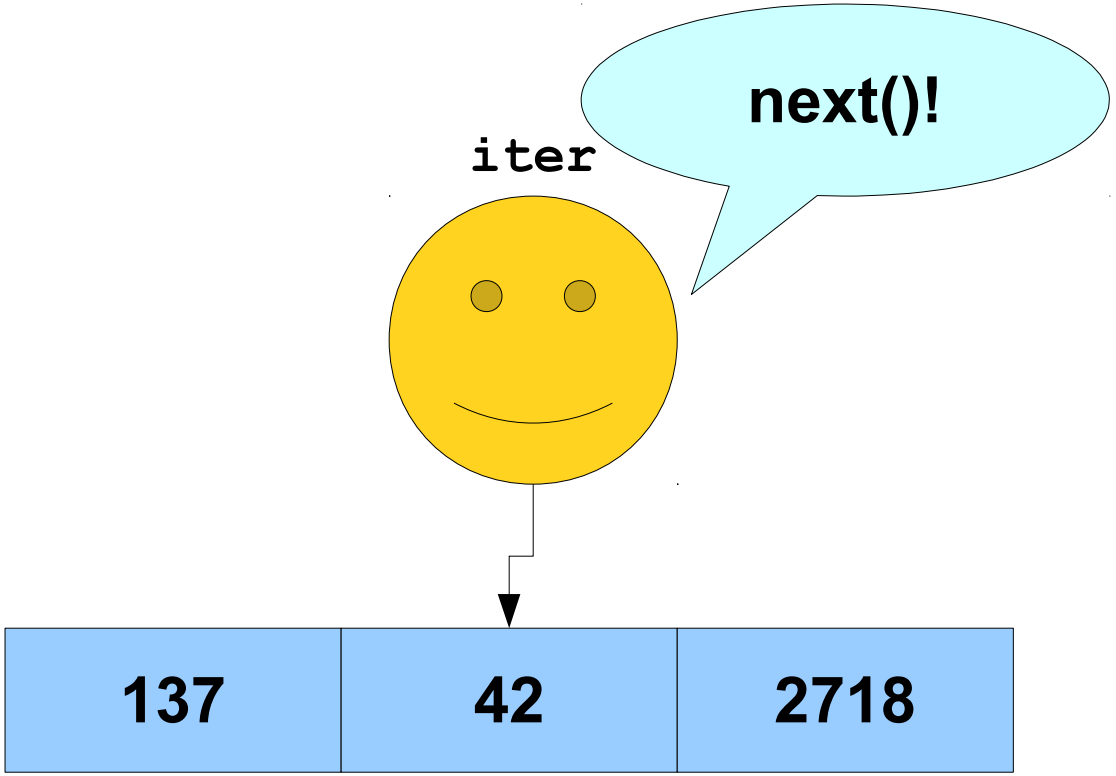



```
ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    int curr = iter.next();

    /* … use curr … */
}
```

Java Iterators

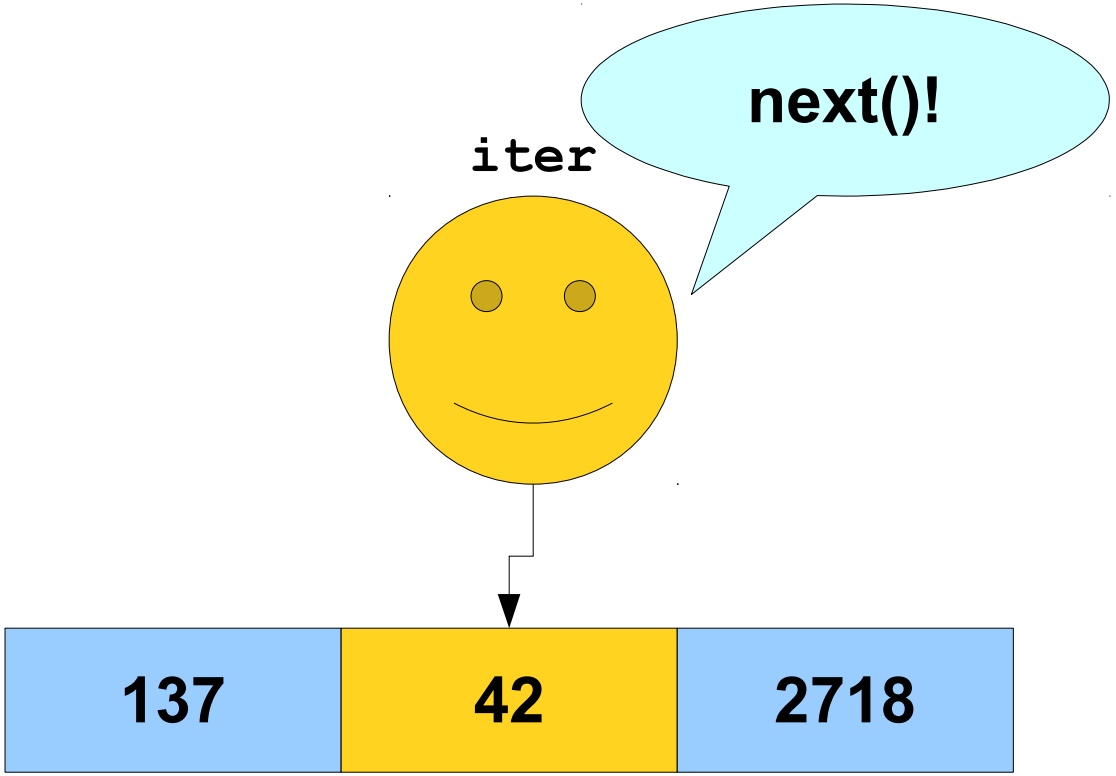



```
ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    int curr = iter.next();

    /* … use curr … */
}
```

Java Iterators

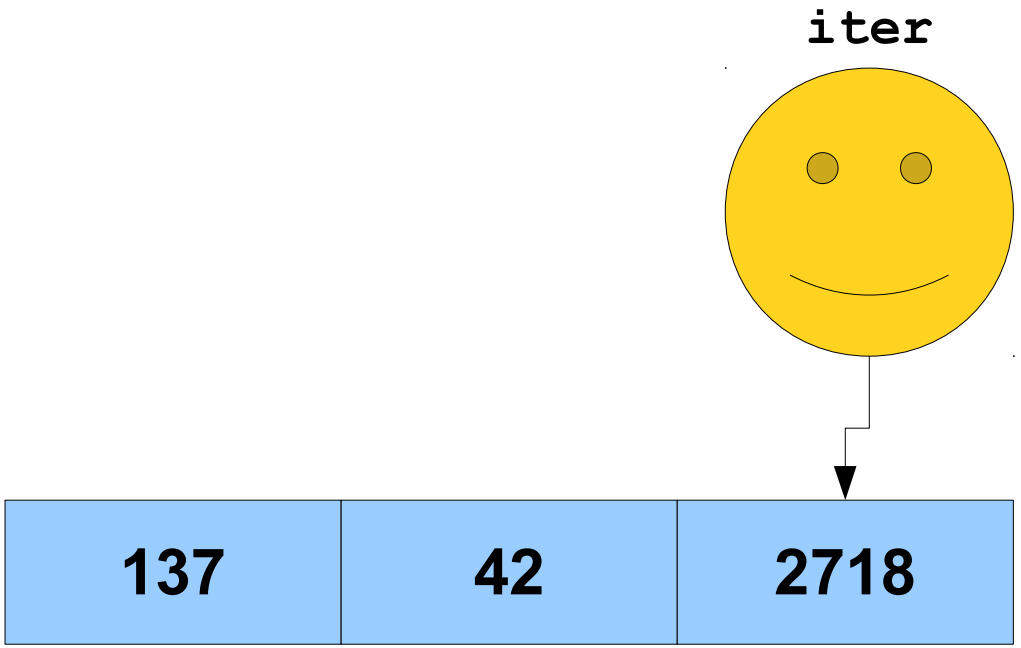



```
ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    int curr = iter.next();

    /* … use curr … */
}
```

Java Iterators

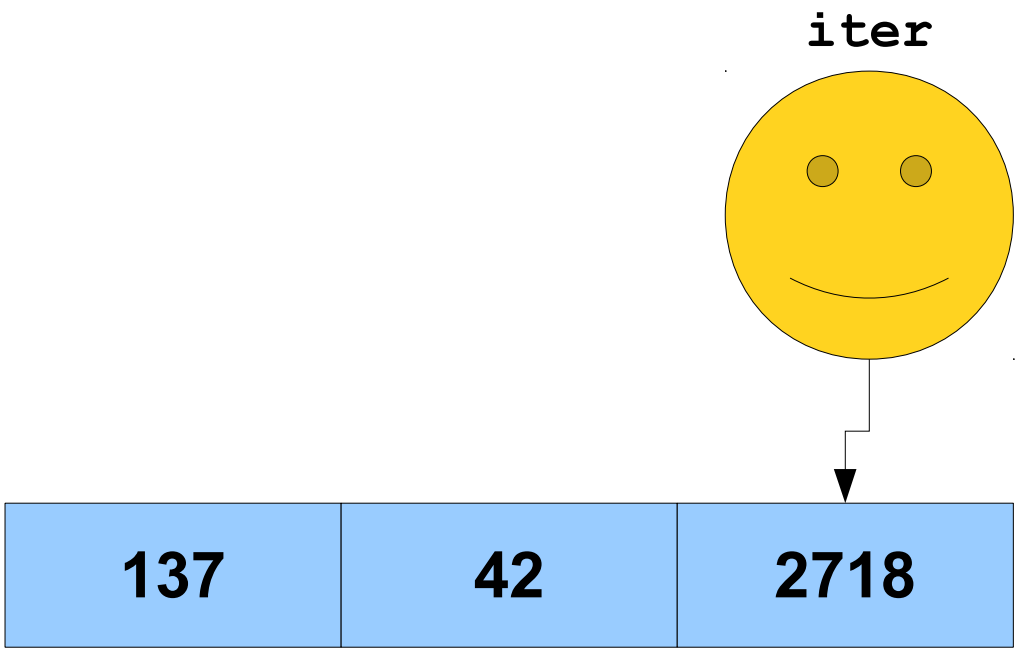



```
ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    int curr = iter.next();

    /* … use curr … */
}
```

Java Iterators

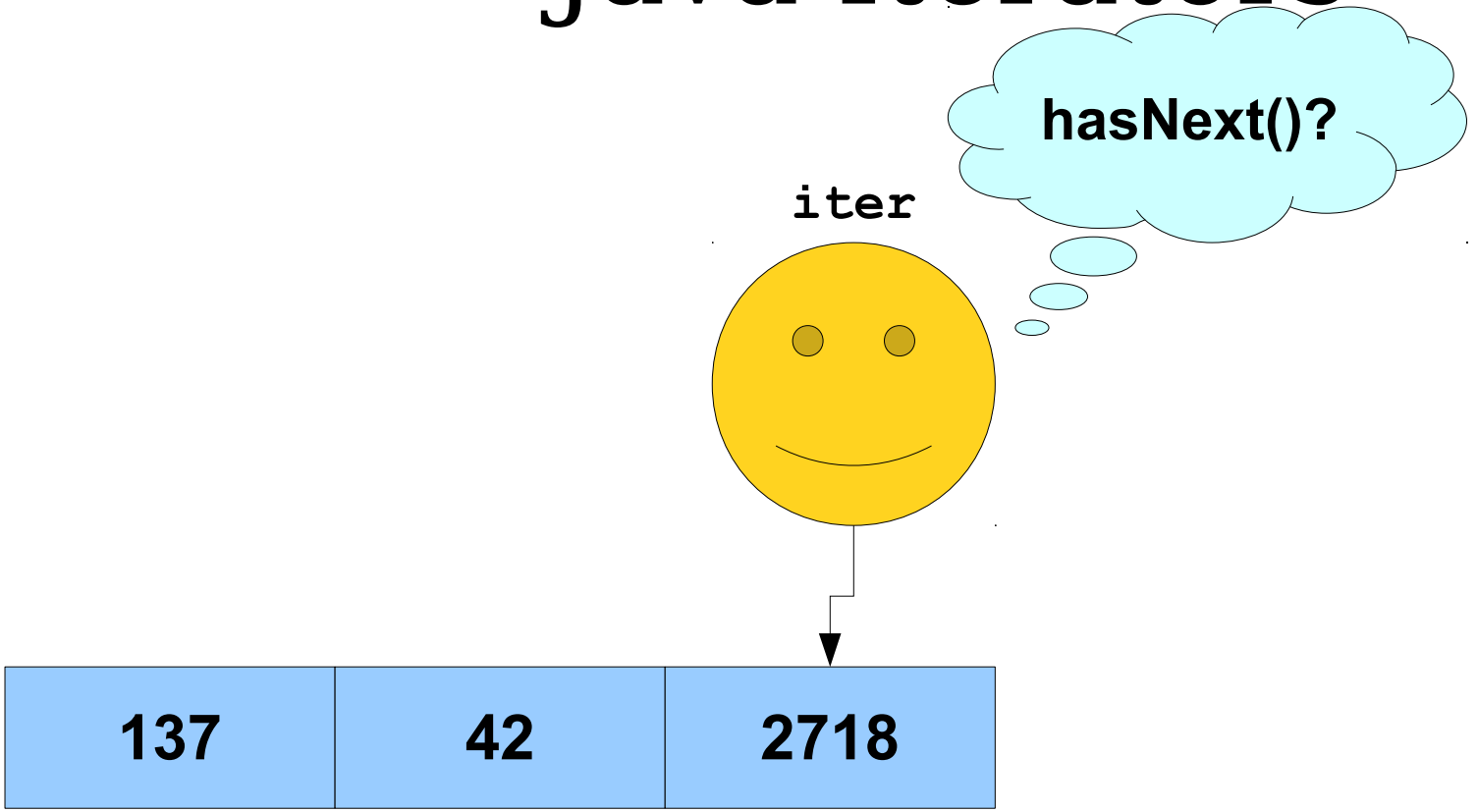



```
ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    int curr = iter.next();

    /* … use curr … */
}
```

Java Iterators

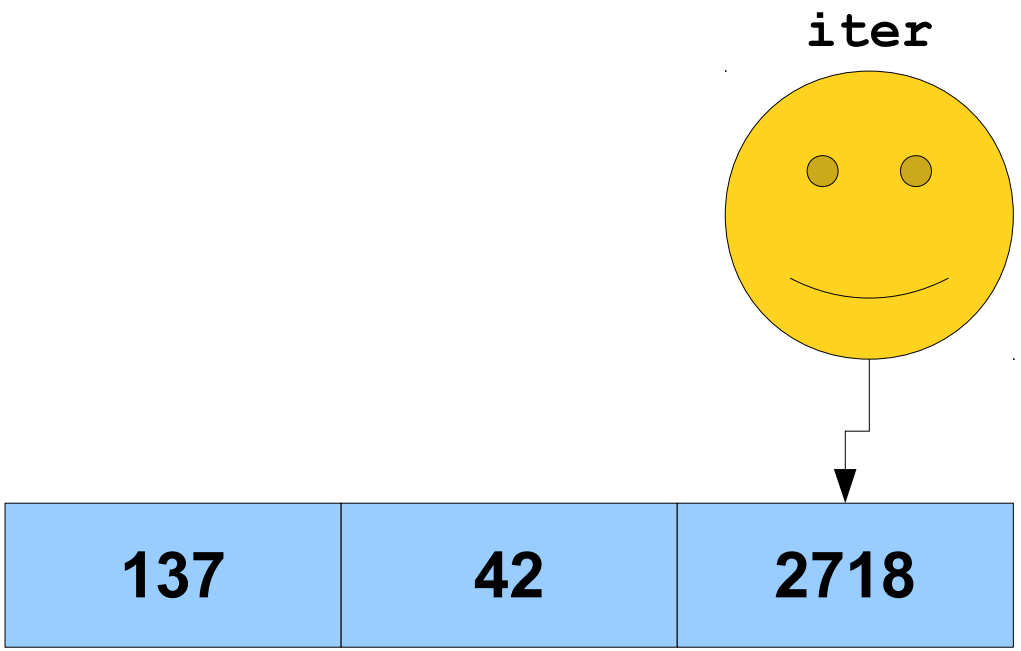



```
ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    int curr = iter.next();

    /* … use curr … */
}
```

Java Iterators

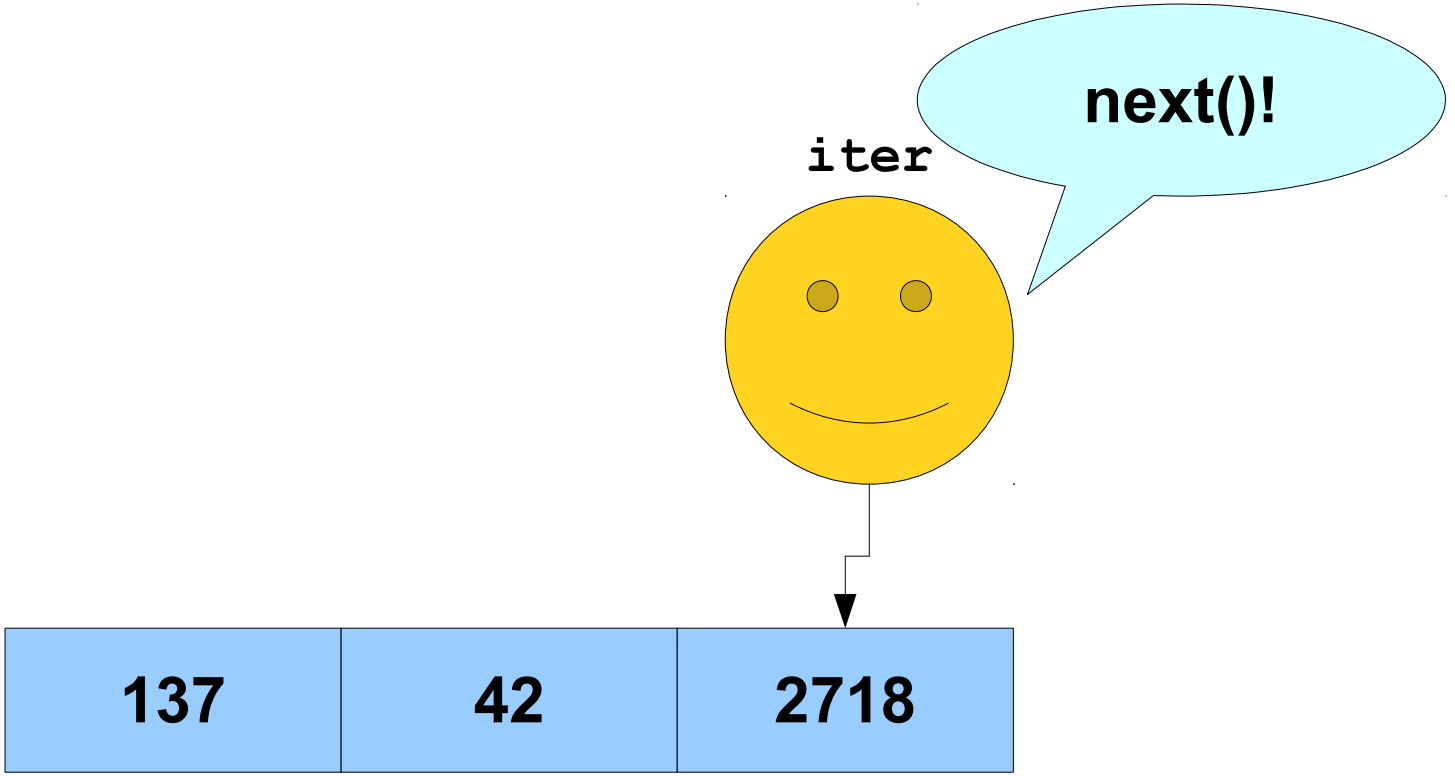



```
ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    int curr = iter.next();

    /* … use curr … */
}
```

Java Iterators

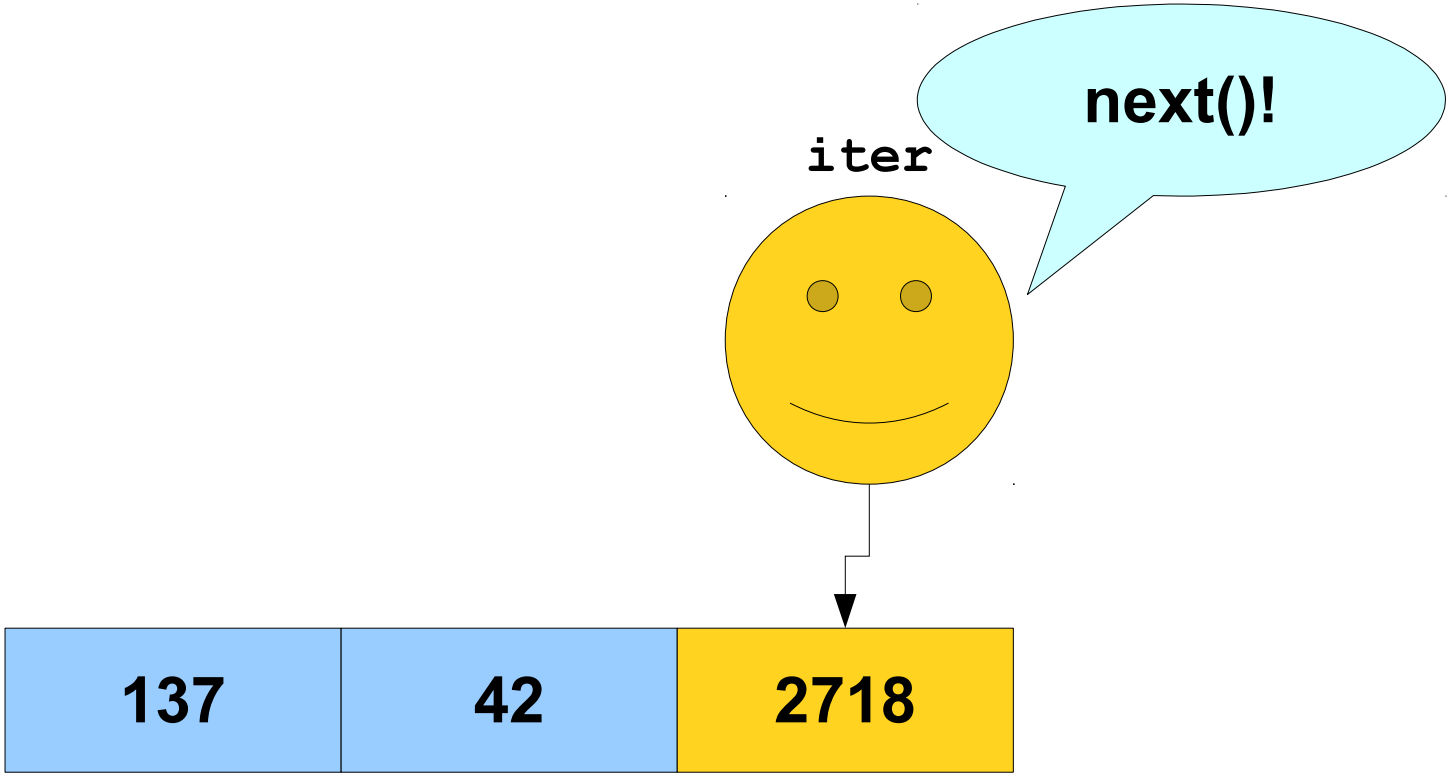



```
ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    int curr = iter.next();

    /* … use curr … */
}
```

Java Iterators

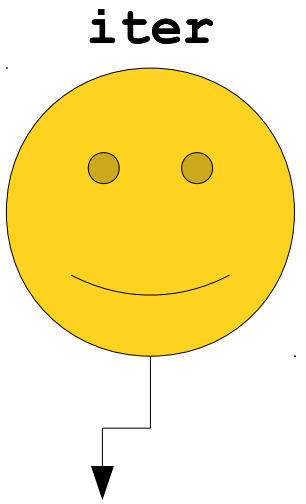



```
ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    int curr = iter.next();

    /* … use curr … */
}
```

Java Iterators



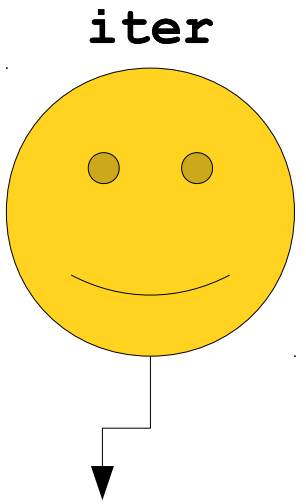
137	42	2718

```
ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    int curr = iter.next();

    /* … use curr … */
}
```

Java Iterators



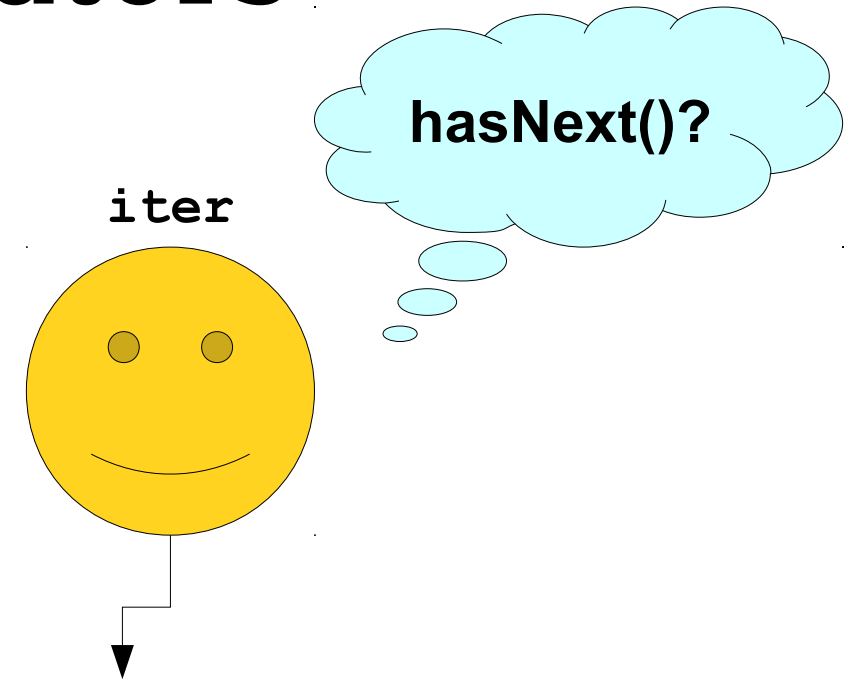
137	42	2718

```
ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    int curr = iter.next();

    /* … use curr … */
}
```

Java Iterators



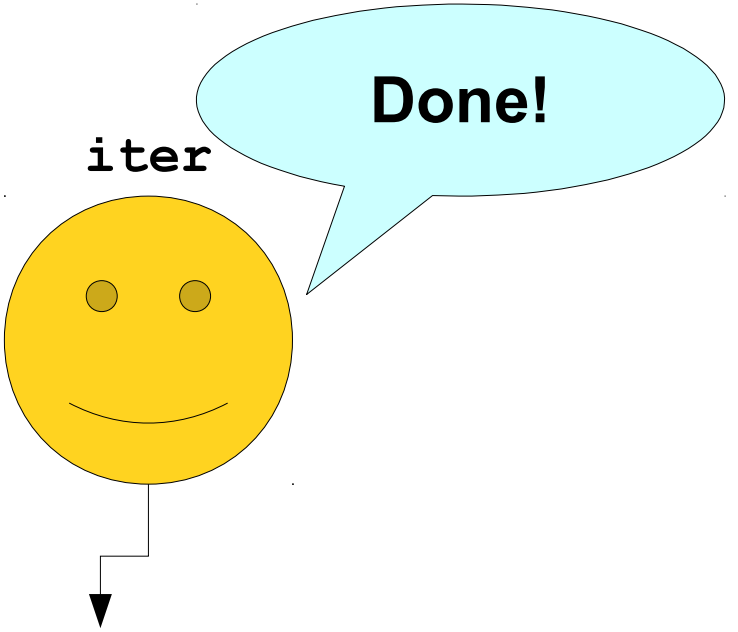
137	42	2718

```
ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    int curr = iter.next();

    /* … use curr … */
}
```

Java Iterators



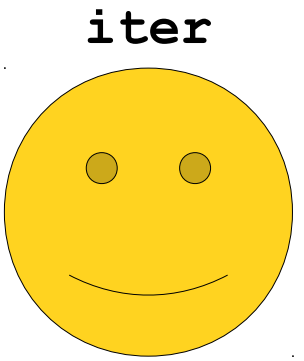
137	42	2718

```
ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    int curr = iter.next();

    /* … use curr … */
}
```

Java Iterators



137	42	2718

```
ArrayList<Integer> myList = /* … */

Iterator<Integer> iter = myList.iterator();
while (iter.hasNext()) {
    int curr = iter.next();

    /* … use curr … */
}
```

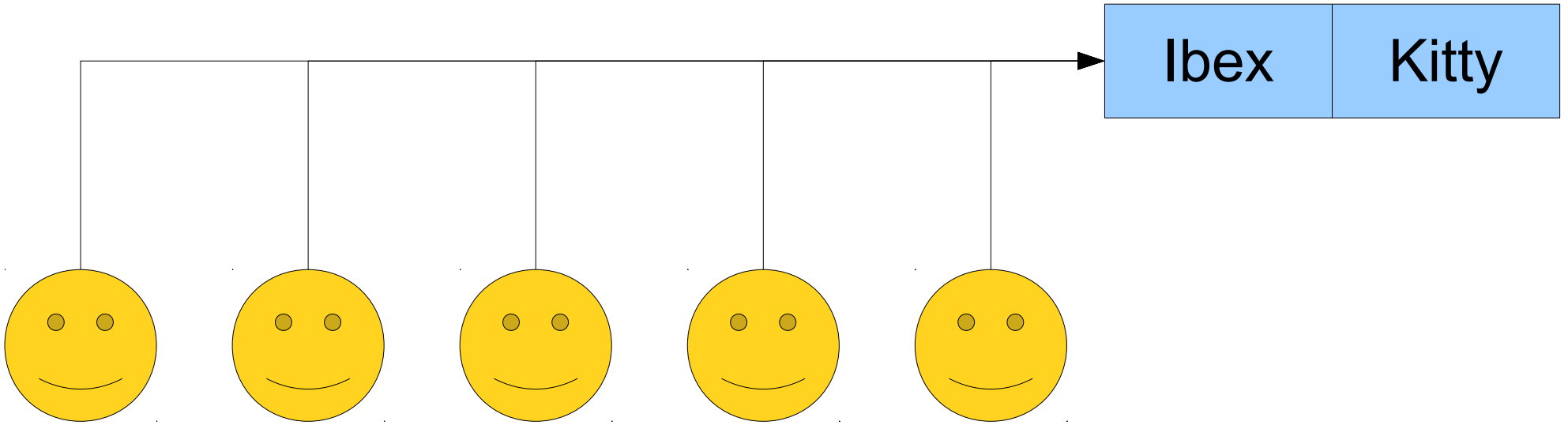
A Word of Warning

A Word of Warning

- The following will loop forever on a nonempty collection:

```
while (collection.iterator().hasNext()) {
    /* … */
}
```

- Every time that you call `.iterator()`, you get back a new iterator to the start of the collection.



A Word of Warning

- The follo collection:

```
while
    /*
}
```

- Every tim ck a new iterator t

ex | Kitty