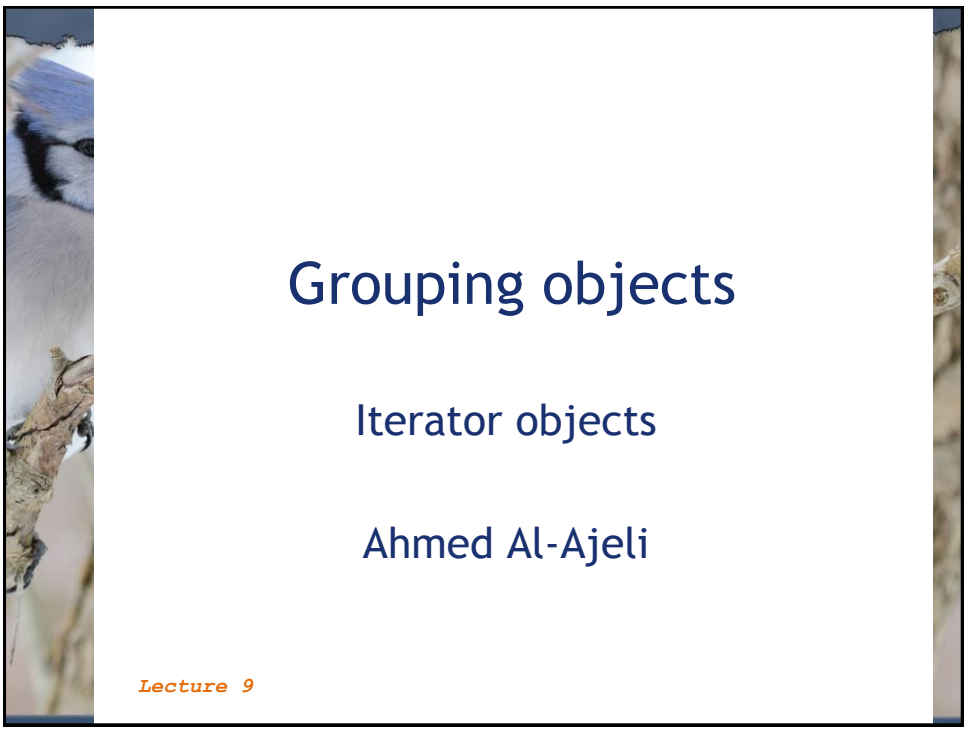
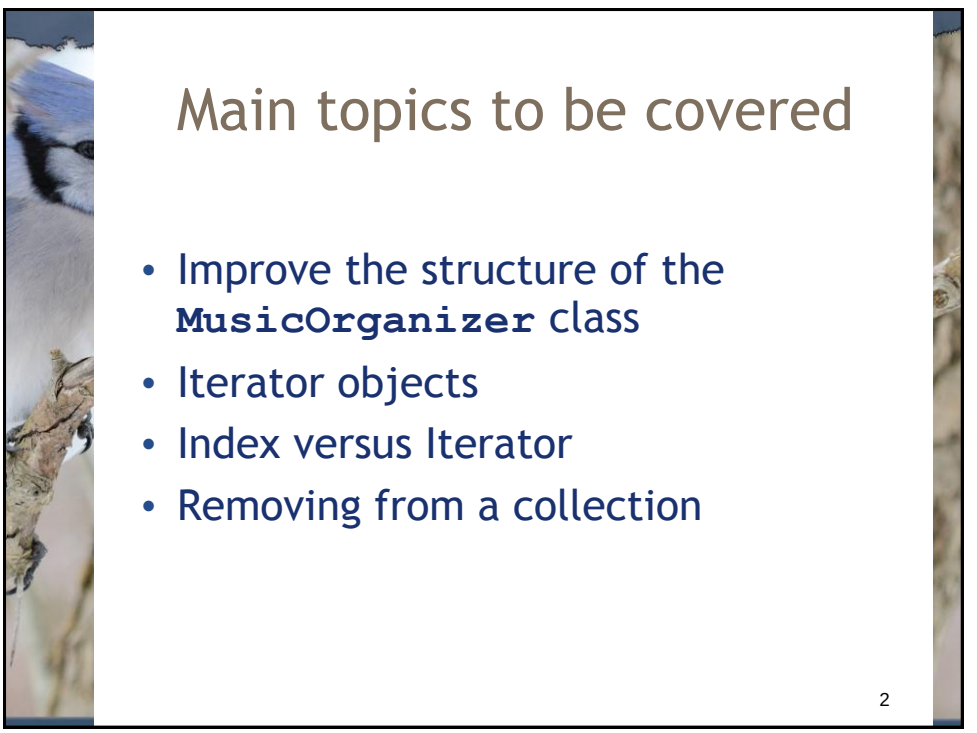


Grouping objects

Iterator objects

Ahmed Al-Ajeli

Lecture 9



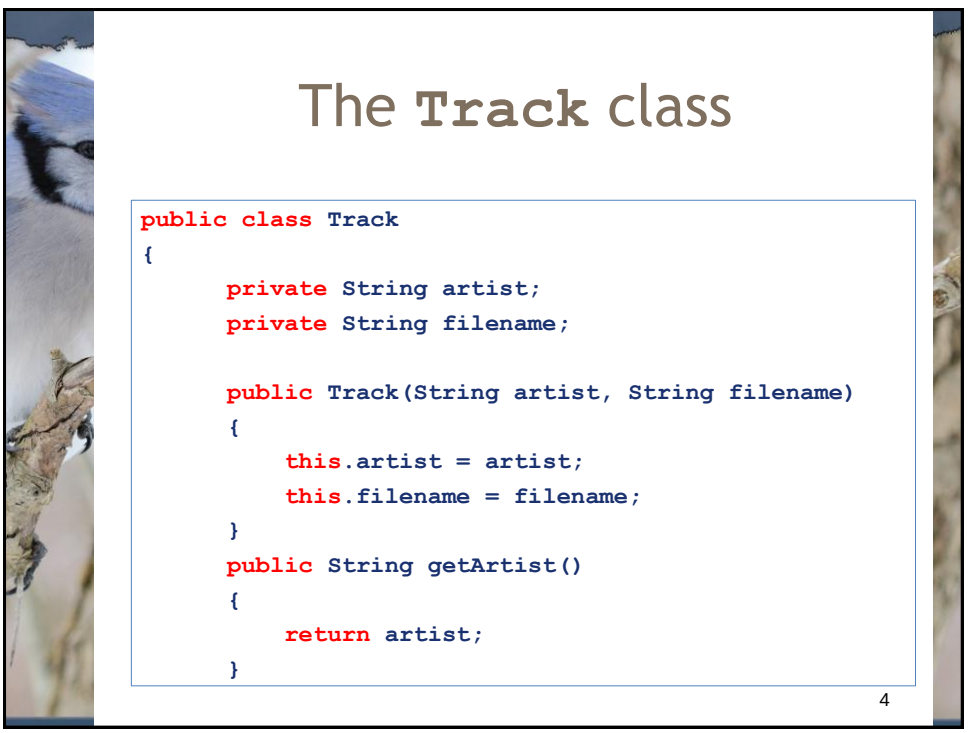
Main topics to be covered

- Improve the structure of the **`MusicOrganizer`** class
- Iterator objects
- Index versus Iterator
- Removing from a collection

Moving away from String

- Our collection of String objects for music tracks is limited.
- No separate identification of artist, title, etc.
- We will define a **`Track`** class with separate fields:
  - **`artist`**
  - **`filename`**

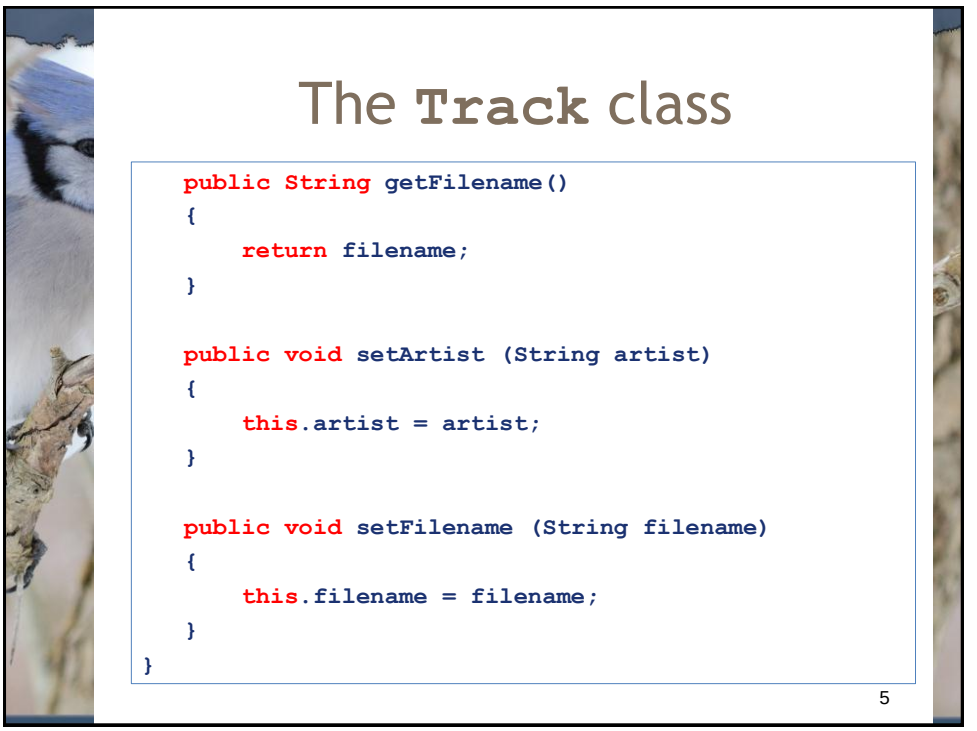
3



The `Track` class

```
public class Track
{
      private String artist;
      private String filename;

      public Track(String artist, String filename)
      {
          this.artist = artist;
          this.filename = filename;
      }
      public String getArtist()
      {
          return artist;
      }
```

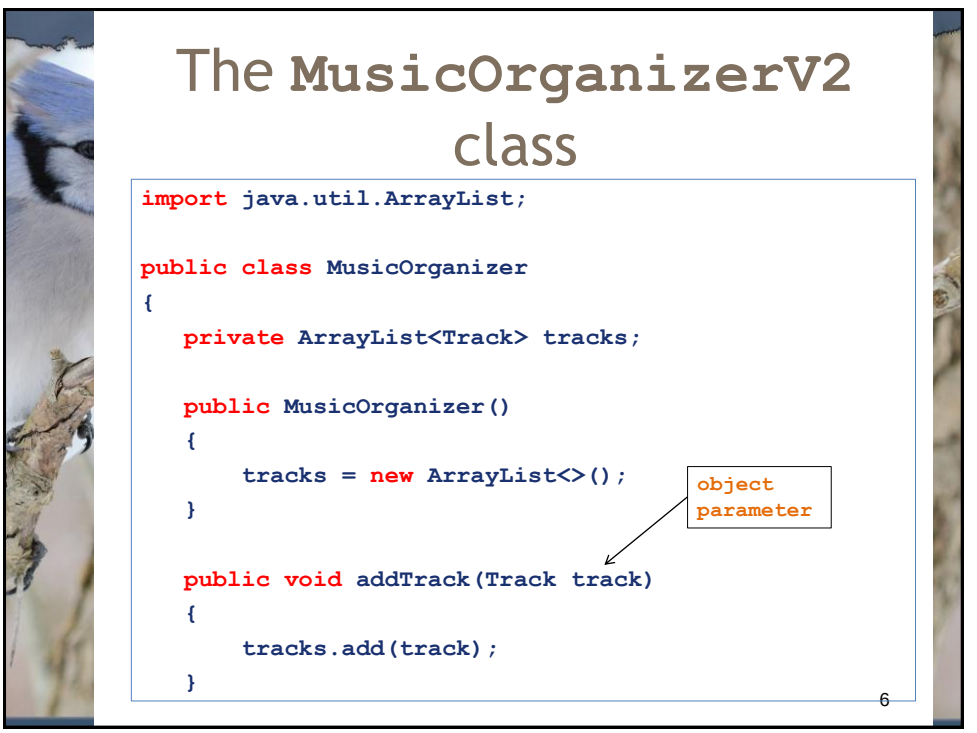
4



The `Track` class

```
    public String getFilename()
    {
        return filename;
    }

    public void setArtist (String artist)
    {
        this.artist = artist;
    }

    public void setFilename (String filename)
    {
        this.filename = filename;
    }
}
```

5



The `MusicOrganizerV2` class

```
import java.util.ArrayList;

public class MusicOrganizer
{
    private ArrayList<Track> tracks;

    public MusicOrganizer()
    {
        tracks = new ArrayList<>();
    }

    public void addTrack(Track track)
    {
        tracks.add(track);
    }
```

object parameter

6

The MusicOrganizerV2 class

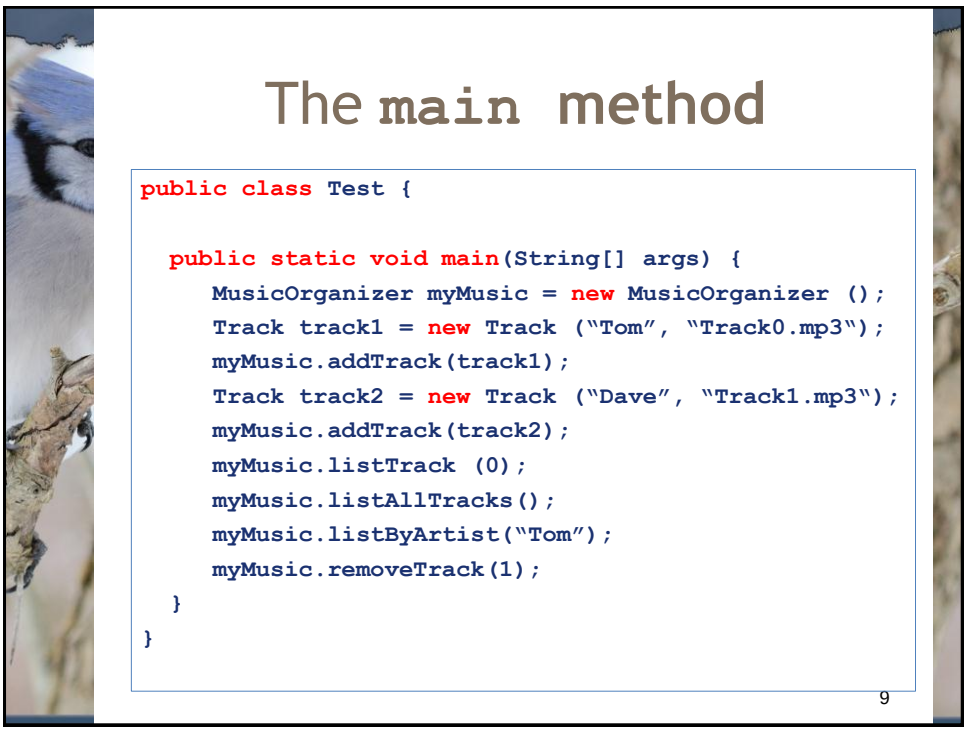
```
public void listTrack(int index)
{
    Track track = tracks.get(index);
    System.out.println("Artist: " +
                track.getArtist()+ "  Filename: " +
                track.getFilename());
}

public void listAllTracks()
{
    for(Track track : tracks) {
        System.out.println("Artist: " +
                track.getArtist()+ "  Filename: " +
                track.getFilename());
   }
}
```

7

The MusicOrganizerV2 class

```
public void listByArtist(String artist)
{
   for(Track track : tracks) {
      if(track.getArtist().contains(artist)) {
         System.out.println("Artist: " +
         track.getArtist()+ "  Filename: " +
         track.getFilename());
      }
   }
}
public void removeTrack(int index)
{
    if(index >= 0 && index < tracks.size()) {
      tracks.remove(index);
    }
}
}
```

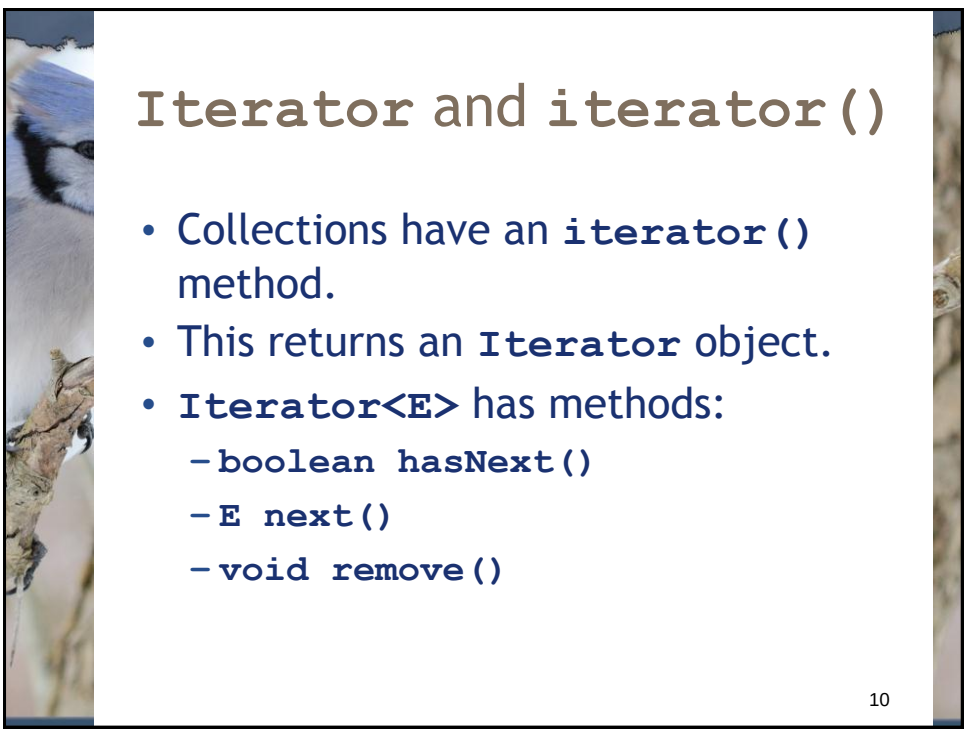
8



The `main` method

```
public class Test {

  public static void main(String[] args) {
     MusicOrganizer myMusic = new MusicOrganizer ();
     Track track1 = new Track ("Tom", "Track0.mp3");
     myMusic.addTrack(track1);
     Track track2 = new Track ("Dave", "Track1.mp3");
     myMusic.addTrack(track2);
     myMusic.listTrack (0);
     myMusic.listAllTracks();
     myMusic.listByArtist("Tom");
     myMusic.removeTrack(1);
  }
}
```

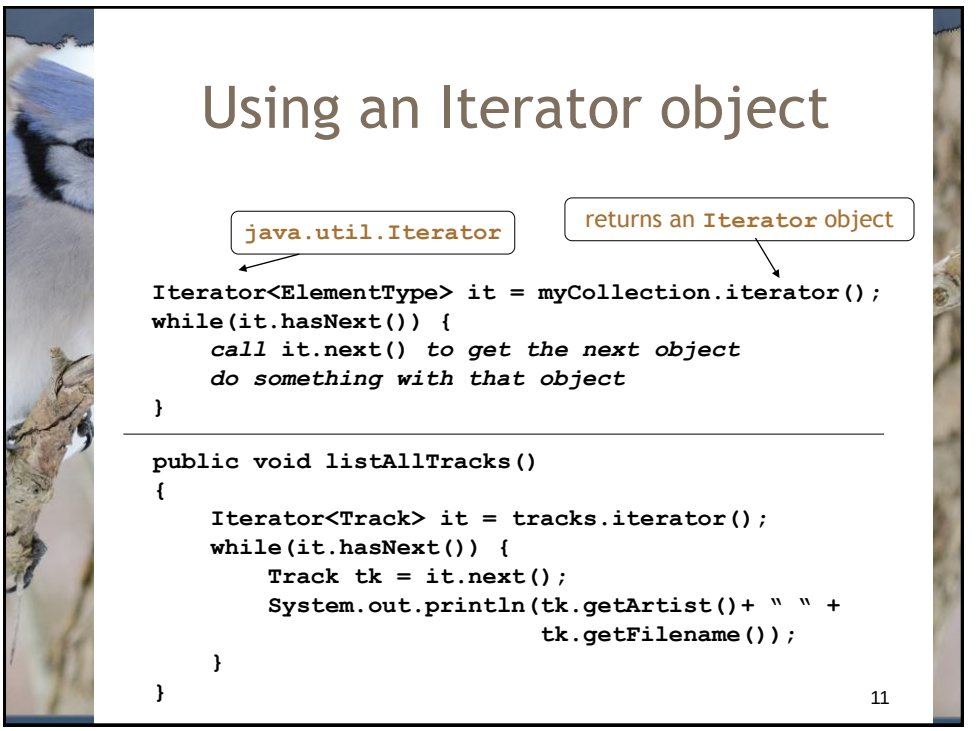
9



`Iterator` and `iterator()`

- Collections have an `iterator()` method.
- This returns an `Iterator` object.
- `Iterator<E>` has methods:
  - `boolean hasNext()`
  - `E next()`
  - `void remove()`

10



Using an Iterator object

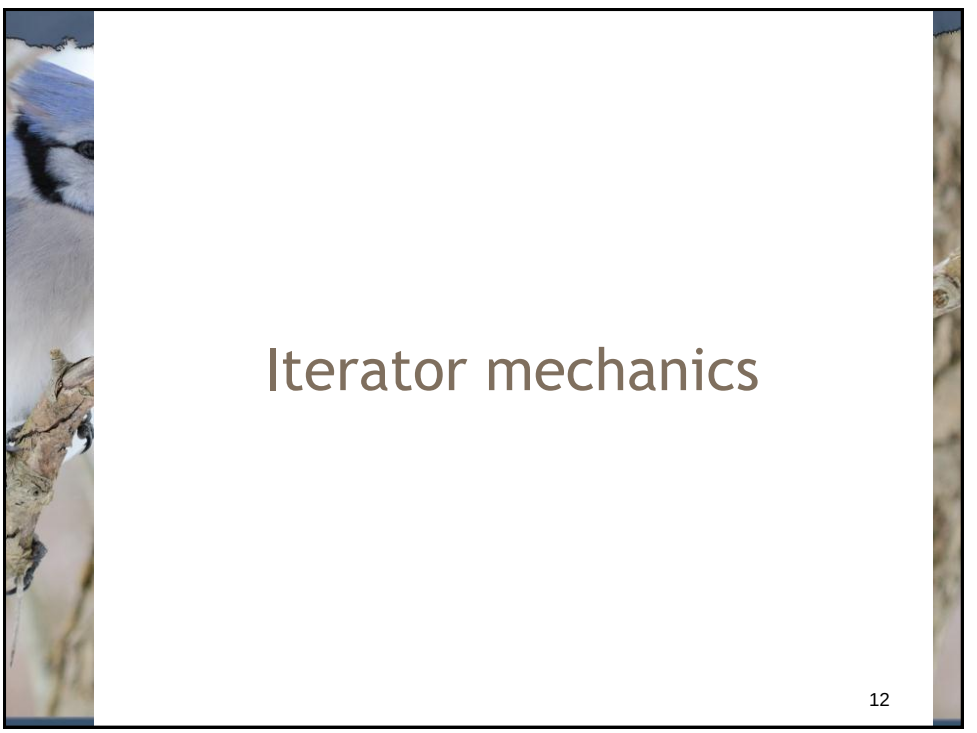
`java.util.Iterator`

returns an `Iterator` object

```
Iterator<ElementType> it = myCollection.iterator();
while(it.hasNext()) {
    call it.next() to get the next object
    do something with that object
}
```

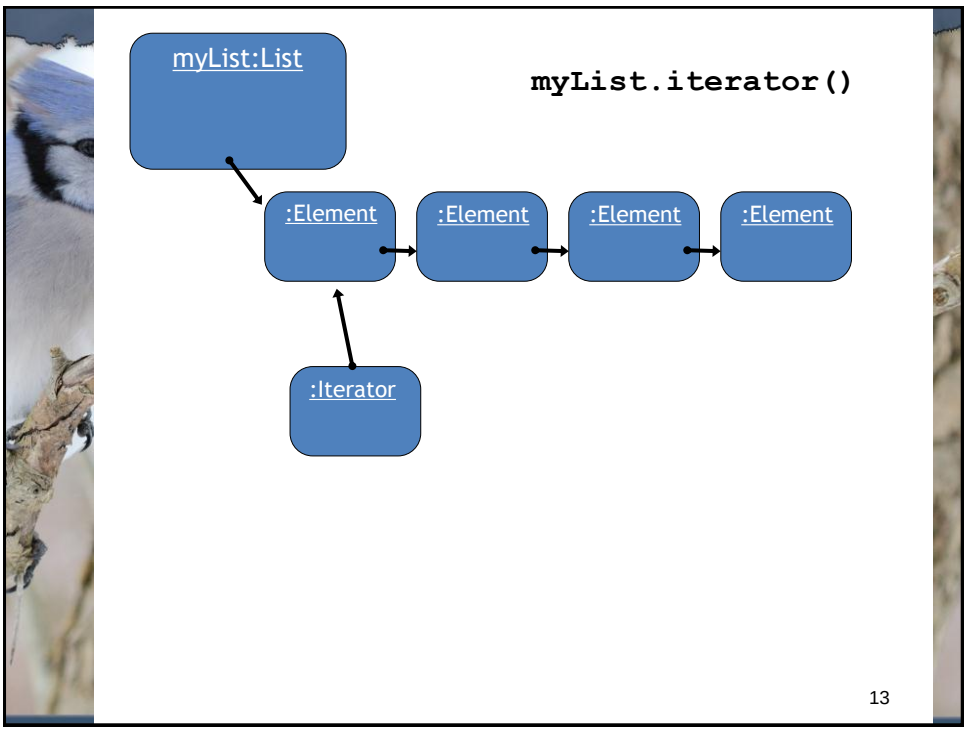
```
public void listAllTracks()
{
    Iterator<Track> it = tracks.iterator();
    while(it.hasNext()) {
        Track tk = it.next();
        System.out.println(tk.getArtist()+ " " +
                           tk.getFilename());
    }
}
```

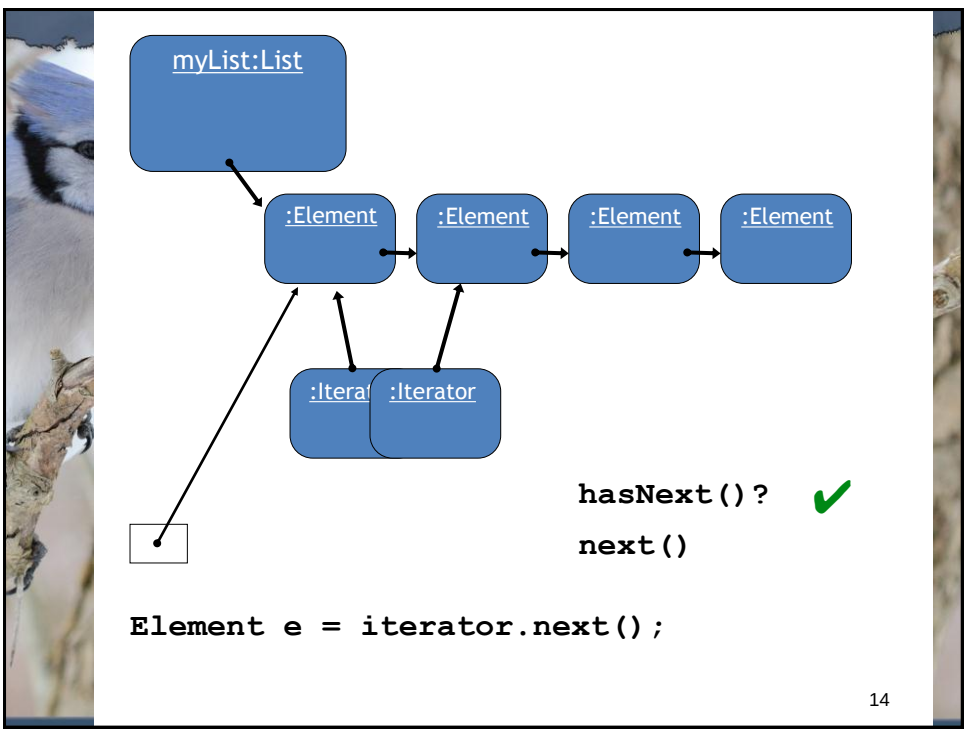
11

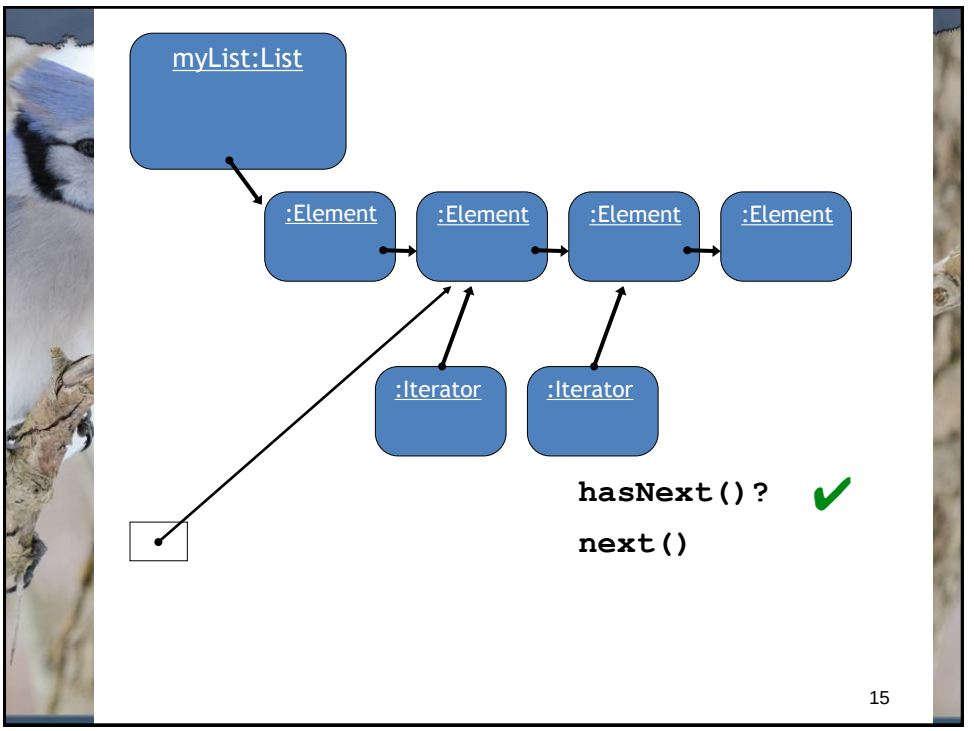


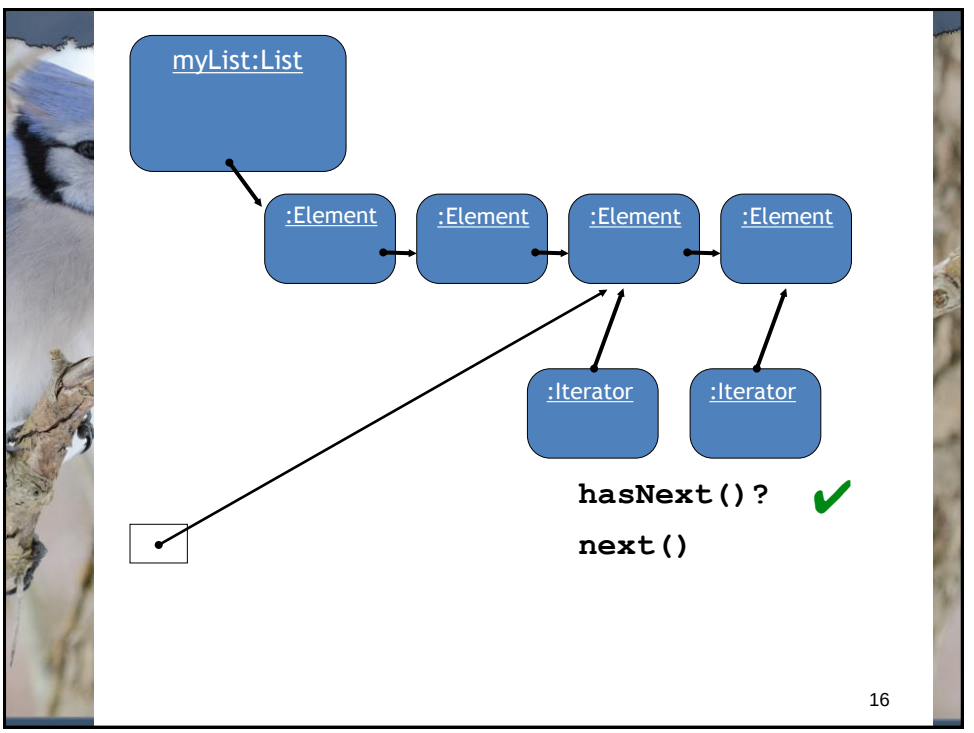
Iterator mechanics

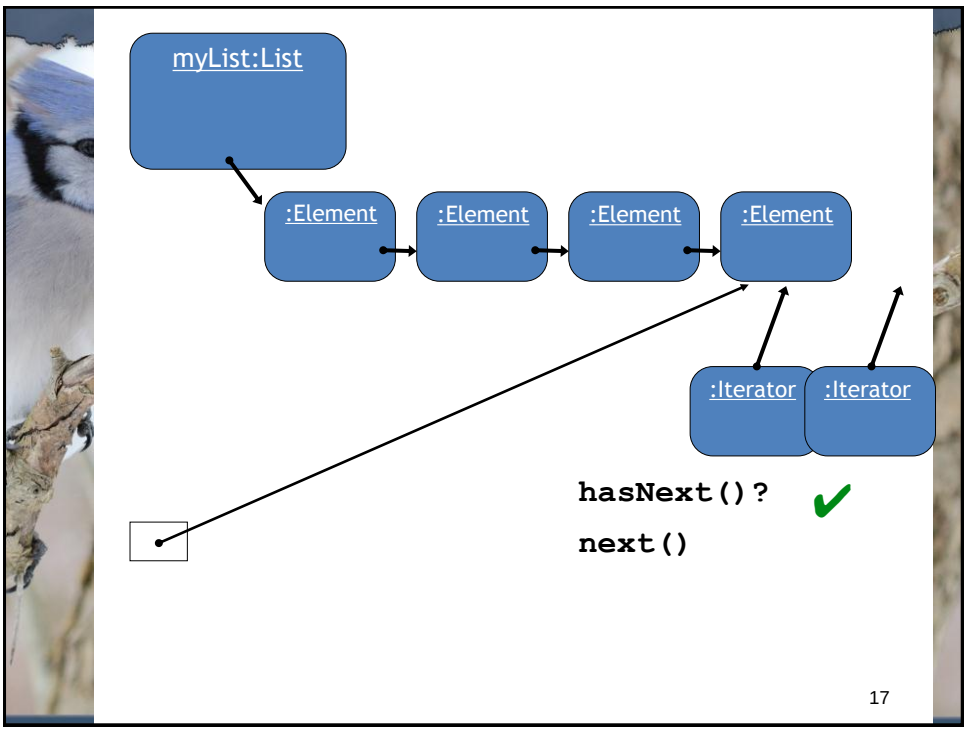
12

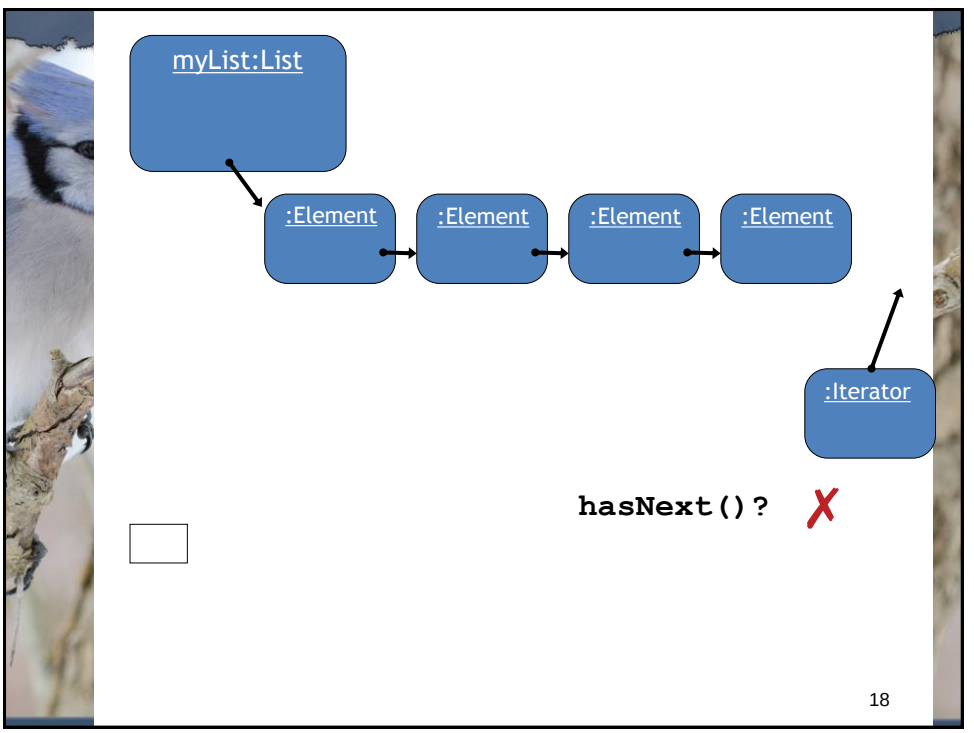

myList:List
myList.iterator()
:Element
:Element
:Element
:Element
:Iterator
13

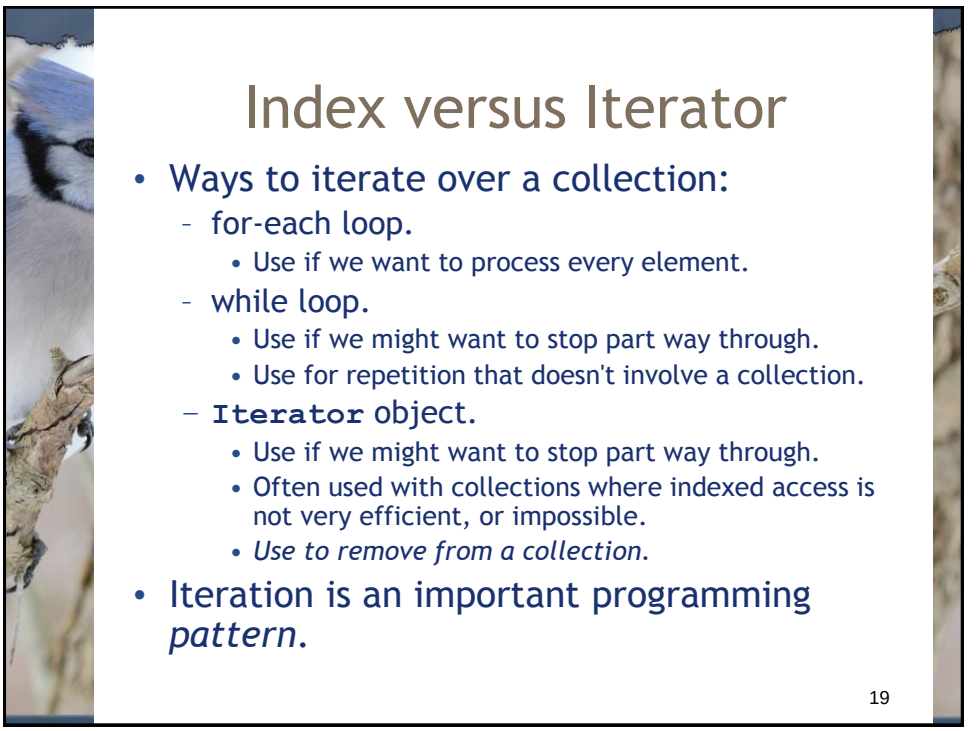

myList:List
:Element
:Element
:Element
:Element
:Iterator
hasNext()?
next()
Element e = iterator.next();
14


myList:List
:Element
:Element
:Element
:Element
:Iterator
:Iterator
hasNext()?
next()
15


myList:List
:Element
:Element
:Element
:Element
:Iterator
:Iterator
hasNext()?
next()
16


myList:List
:Element
:Element
:Element
:Element
:Iterator
:Iterator
hasNext()?
next()
17

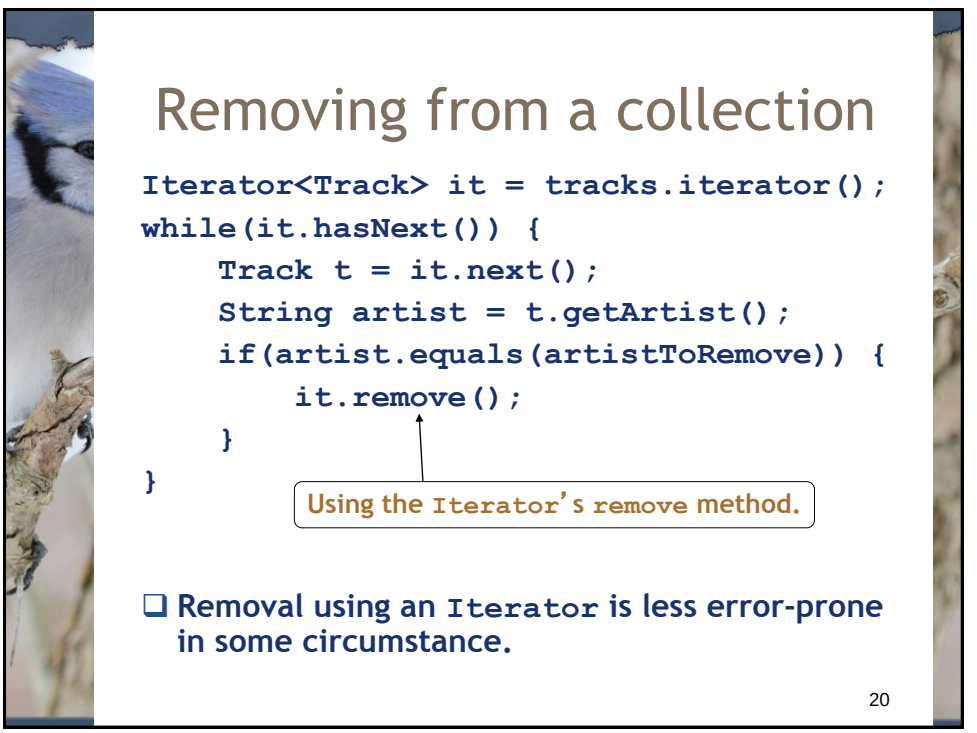

myList:List
:Element
:Element
:Element
:Element
:Iterator
hasNext()?
18



Index versus Iterator

- Ways to iterate over a collection:
  - for-each loop.
    - Use if we want to process every element.
  - while loop.
    - Use if we might want to stop part way through.
    - Use for repetition that doesn't involve a collection.
  - `Iterator` object.
    - Use if we might want to stop part way through.
    - Often used with collections where indexed access is not very efficient, or impossible.
    - *Use to remove from a collection.*
- Iteration is an important programming *pattern*.

19



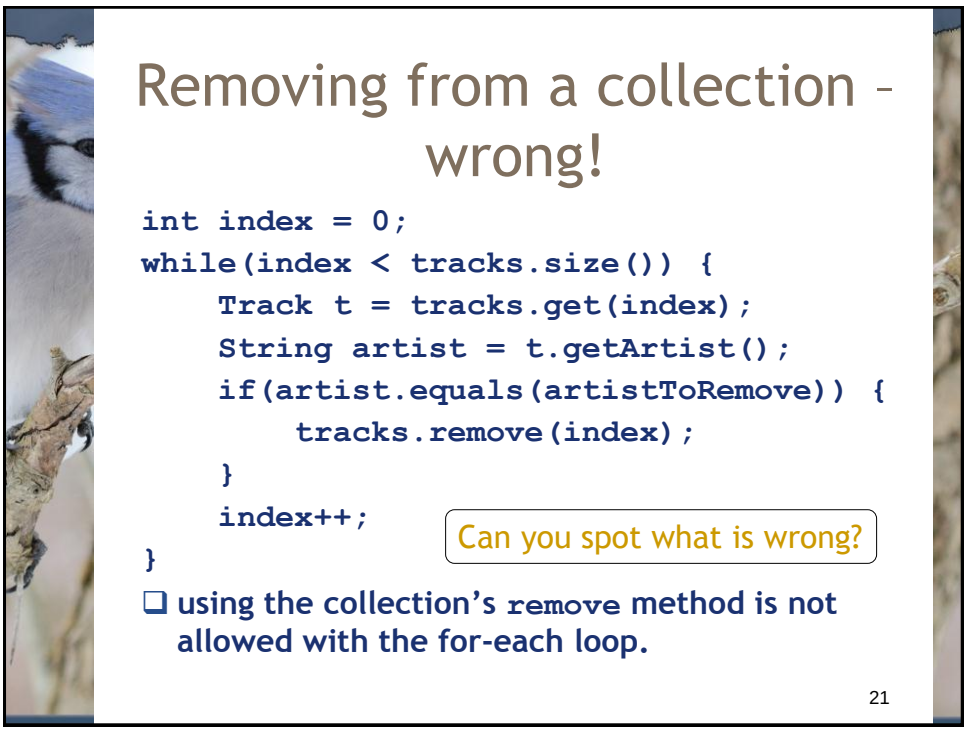
Removing from a collection

```
Iterator<Track> it = tracks.iterator();
while(it.hasNext()) {
    Track t = it.next();
    String artist = t.getArtist();
    if(artist.equals(artistToRemove)) {
        it.remove();
    }
}
```

Using the `Iterator`'s `remove` method.

- Removal using an `Iterator` is less error-prone in some circumstance.

20



Removing from a collection – wrong!

```
int index = 0;
while(index < tracks.size()) {
    Track t = tracks.get(index);
    String artist = t.getArtist();
    if(artist.equals(artistToRemove)) {
        tracks.remove(index);
    }
    index++;
}
```

Can you spot what is wrong?

- using the collection's `remove` method is not allowed with the for-each loop.

21