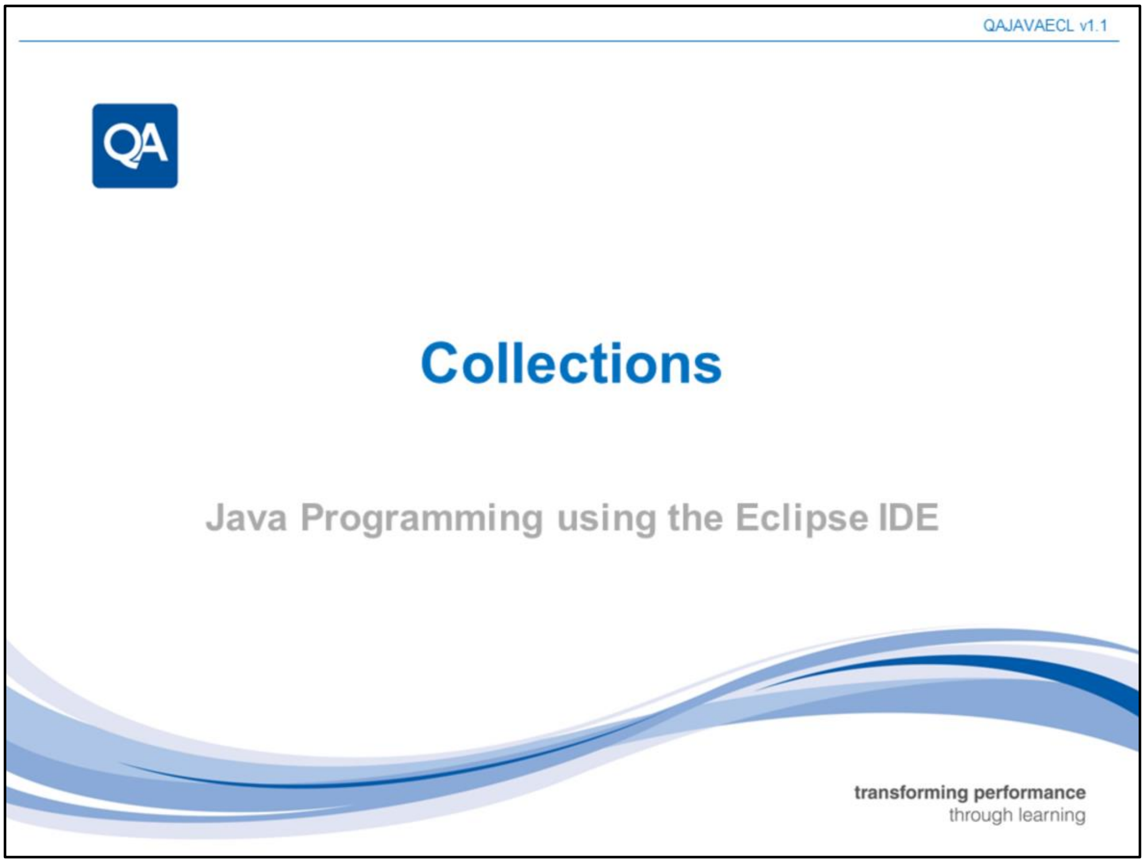
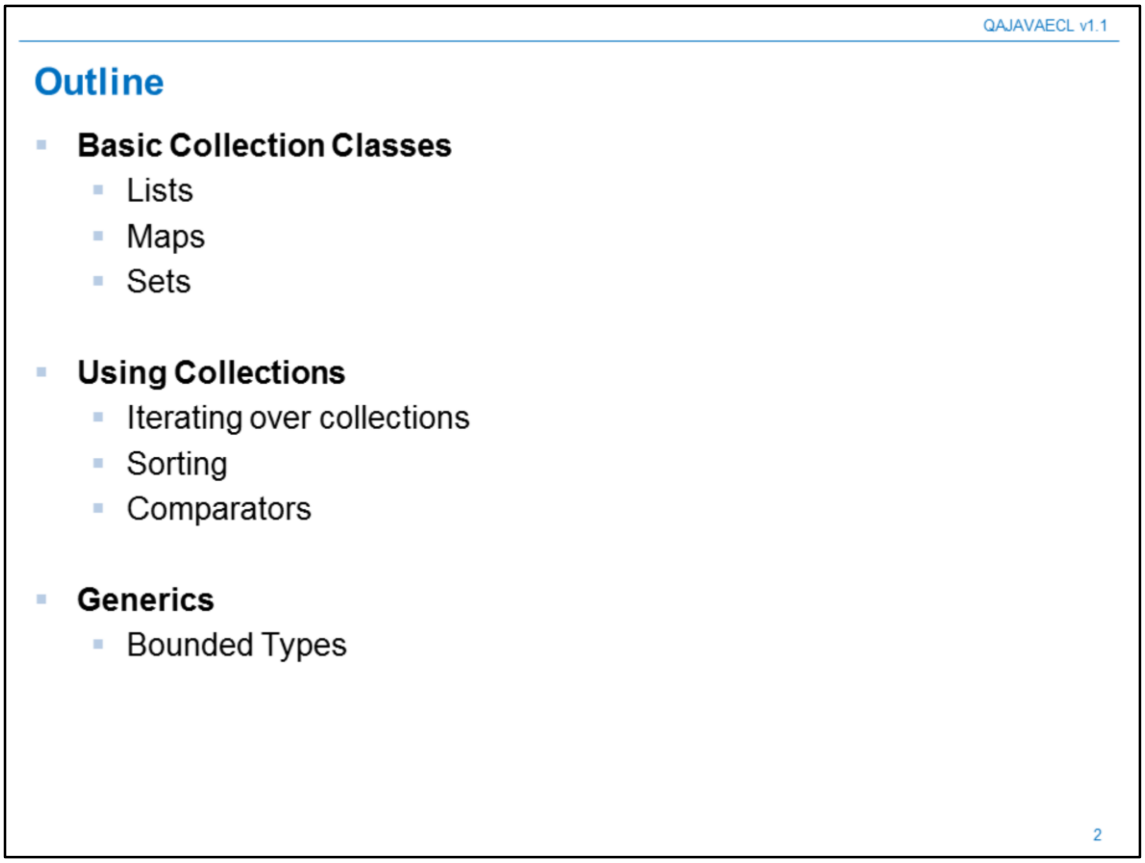


QAJAVAECL v1.1

Collections

Java Programming using the Eclipse IDE

transforming performance
through learning

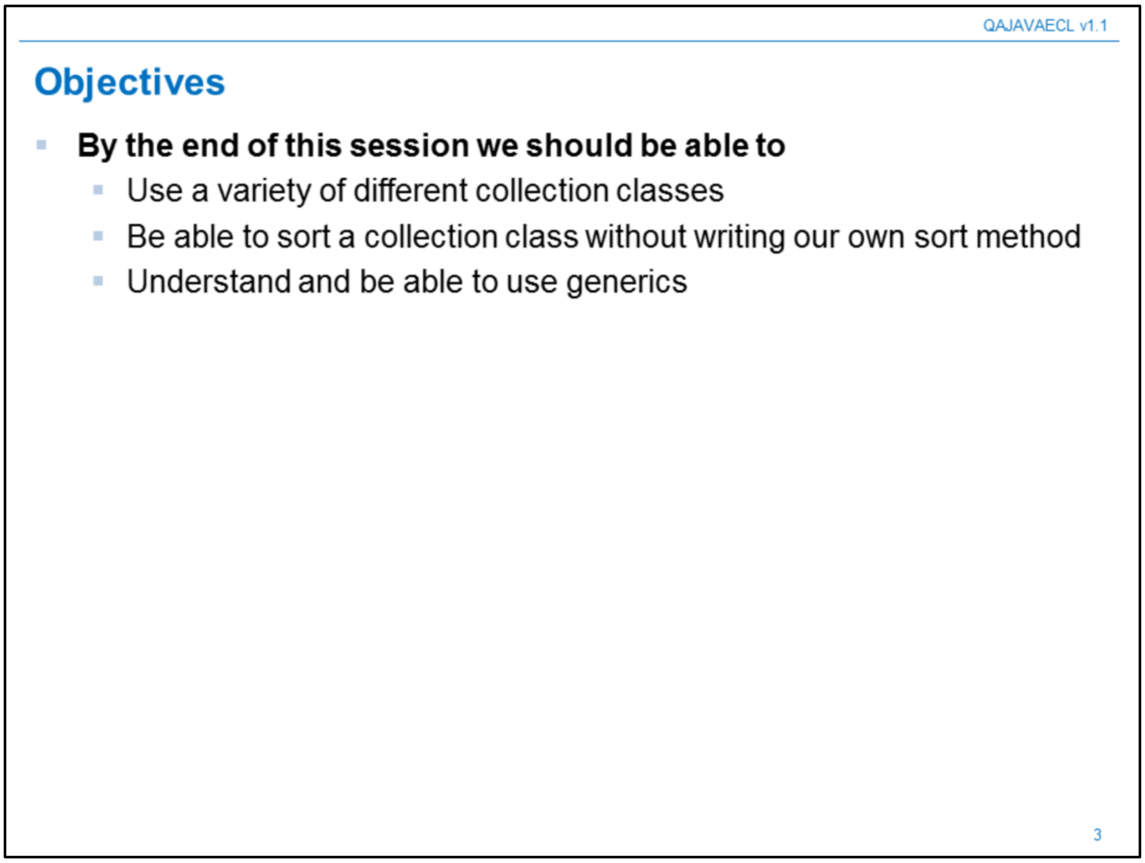


QAJAVAECL v1.1

Outline

- **Basic Collection Classes**
  - Lists
  - Maps
  - Sets
- **Using Collections**
  - Iterating over collections
  - Sorting
  - Comparators
- **Generics**
  - Bounded Types

2

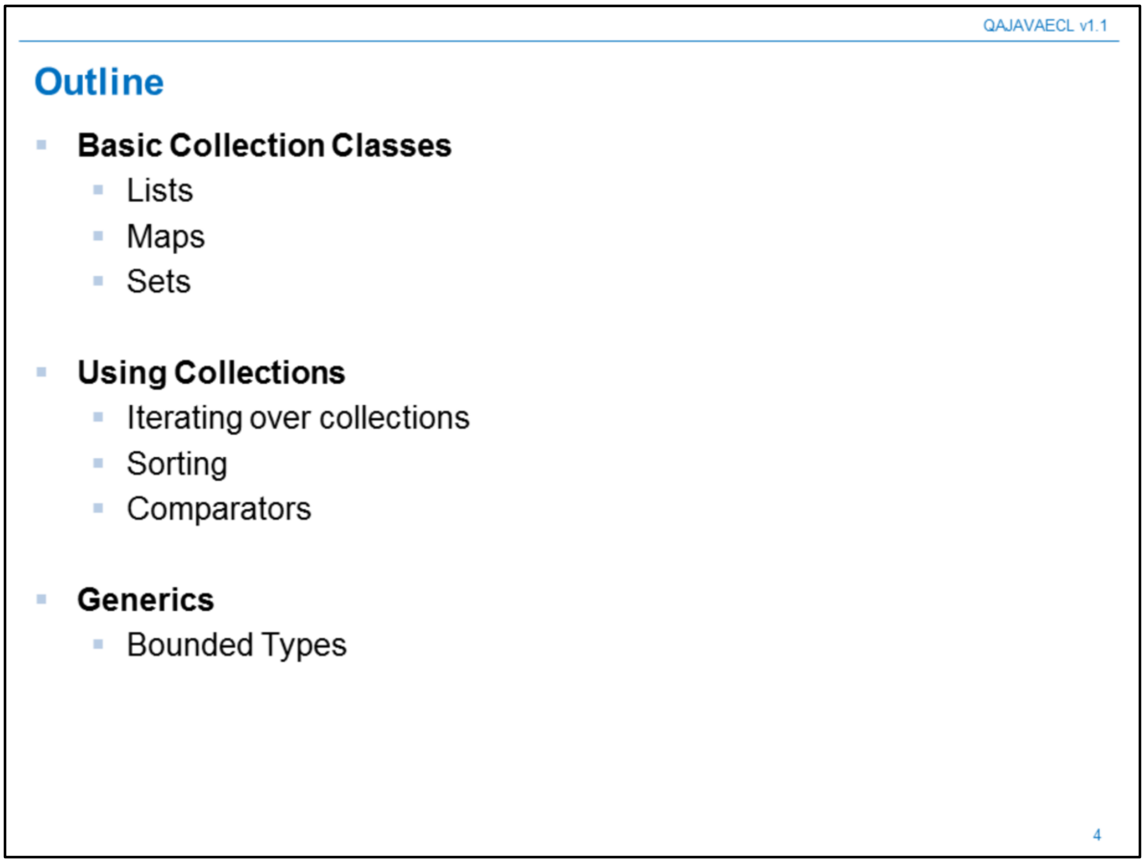


QAJAVAECL v1.1

Objectives

- **By the end of this session we should be able to**
  - Use a variety of different collection classes
  - Be able to sort a collection class without writing our own sort method
  - Understand and be able to use generics

3

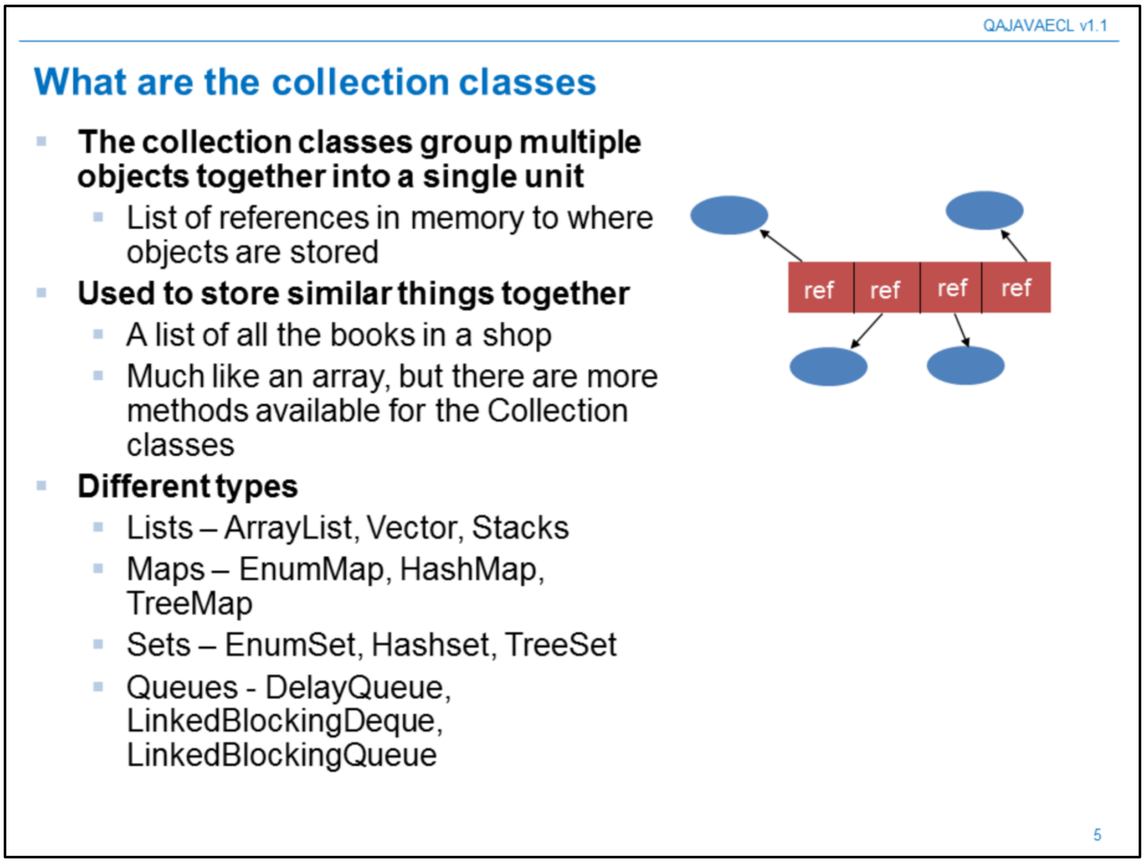


QAJAVAECL v1.1

Outline

- **Basic Collection Classes**
  - Lists
  - Maps
  - Sets
- **Using Collections**
  - Iterating over collections
  - Sorting
  - Comparators
- **Generics**
  - Bounded Types

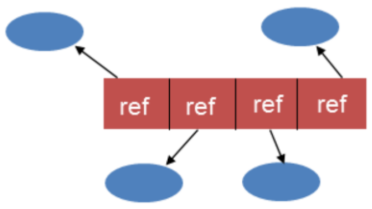
4



QAJAVAECL v1.1

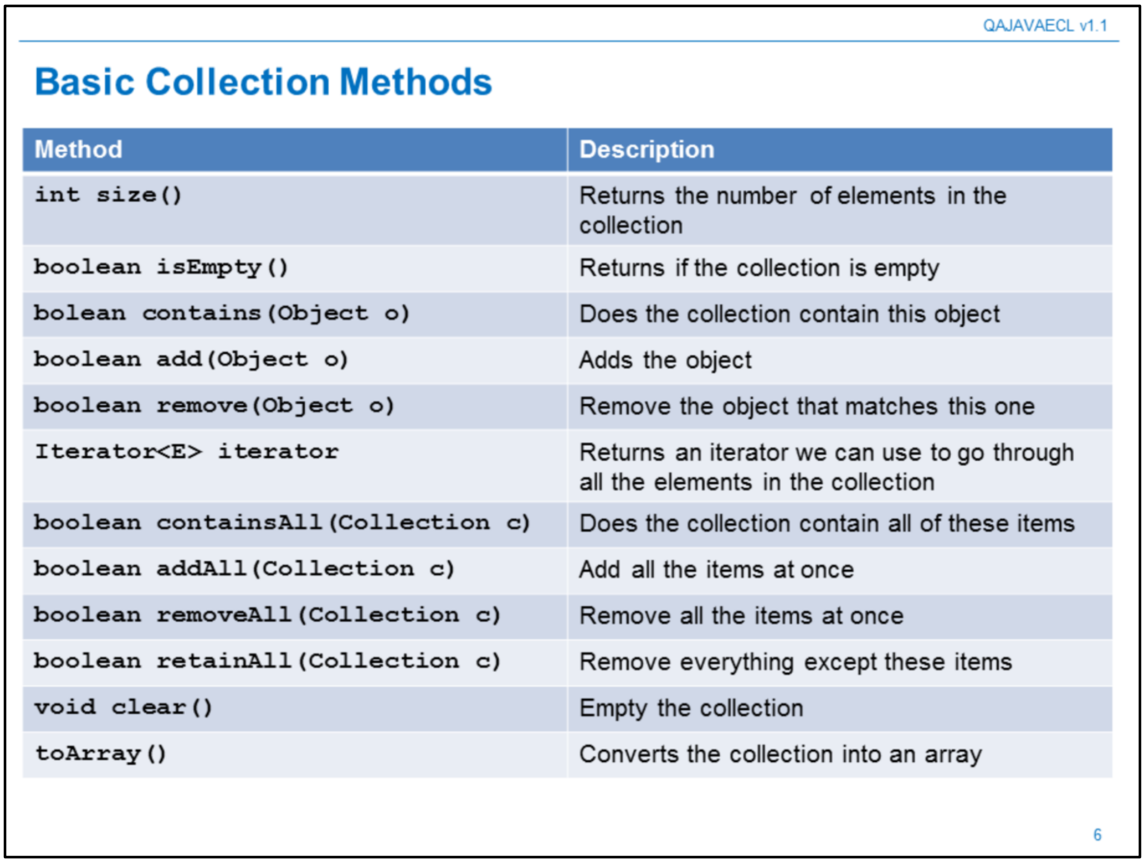
What are the collection classes

- **The collection classes group multiple objects together into a single unit**
  - List of references in memory to where objects are stored
- **Used to store similar things together**
  - A list of all the books in a shop
  - Much like an array, but there are more methods available for the Collection classes
- **Different types**
  - Lists – ArrayList, Vector, Stacks
  - Maps – EnumMap, HashMap, TreeMap
  - Sets – EnumSet, Hashset, TreeSet
  - Queues - DelayQueue, LinkedBlockingDeque, LinkedBlockingQueue



5

This is not an exhaustive list. Map is not quite a collection as it implements a different set of methods, but it is often grouped with the collection classes regardless



QAJAVAECL v1.1

Basic Collection Methods

Method	Description
`int size()`	Returns the number of elements in the collection
`boolean isEmpty()`	Returns if the collection is empty
`bolean contains(Object o)`	Does the collection contain this object
`boolean add(Object o)`	Adds the object
`boolean remove(Object o)`	Remove the object that matches this one
`Iterator<E> iterator`	Returns an iterator we can use to go through all the elements in the collection
`boolean containsAll(Collection c)`	Does the collection contain all of these items
`boolean addAll(Collection c)`	Add all the items at once
`boolean removeAll(Collection c)`	Remove all the items at once
`boolean retainAll(Collection c)`	Remove everything except these items
`void clear()`	Empty the collection
`toArray()`	Converts the collection into an array

6

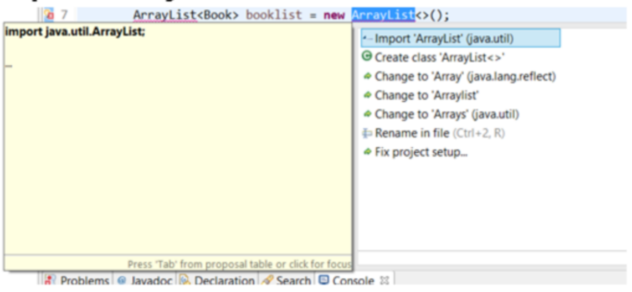
These are all part of the Abstract Collection class which all the collections inherit from. This means that no matter which type of collection we are using, these methods will exist.

Importing the collection classes

- **To use any of the collections we need to import them into the project**
  - The import statement comes after the package statement, but before the class statement
  - This adds the ability to create objects and call methods from classes in other packages

```
import java.util.ArrayList;
```

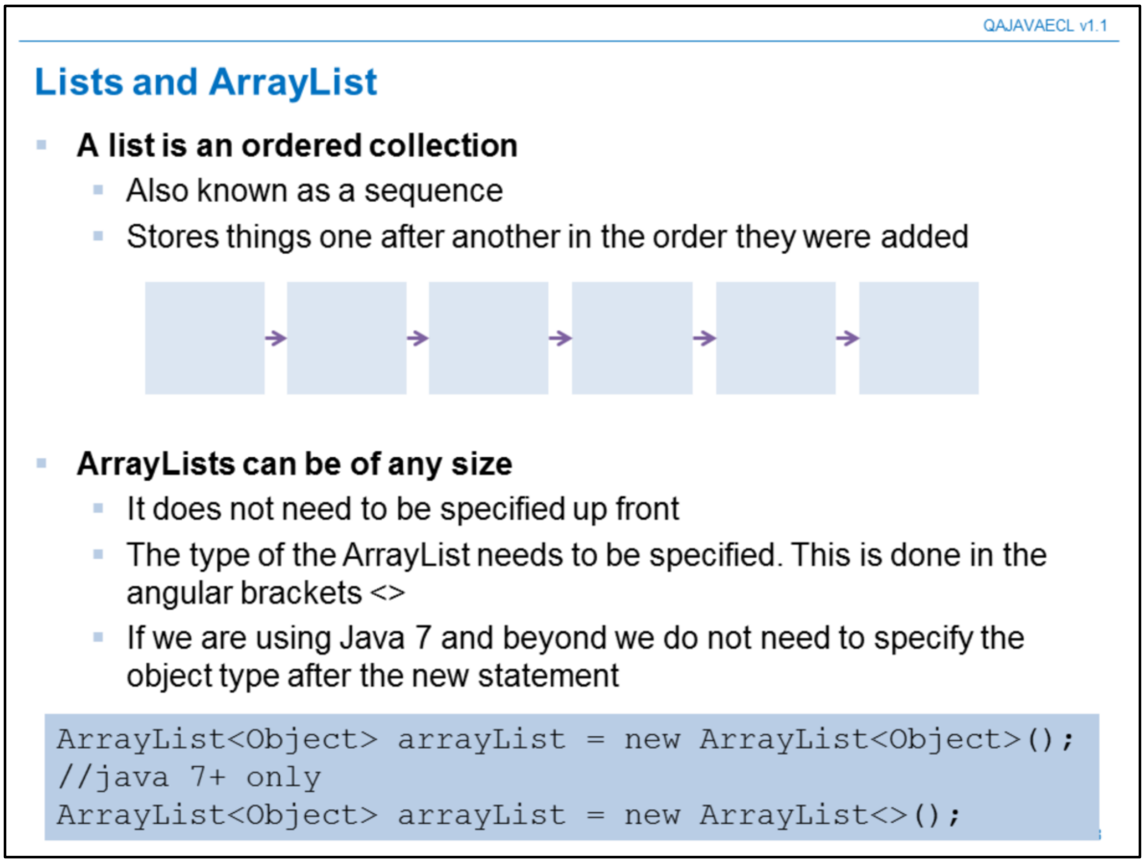
- **Eclipse can sort out the imports for you**
  - Ctrl+Shift+O
  - Context Menu Options



```
ArrayList<Book> booklist = new ArrayList<>();
```

```
import java.util.ArrayList;
```

Import 'ArrayList' (java.util)
Create class 'ArrayList<>'
Change to 'Array' (java.lang.reflect)
Change to 'Arraylist'
Change to 'Arrays' (java.util)
Rename in file (Ctrl+2, R)
Fix project setup...
Press 'Tab' from proposal table or click for focus
Problems
Javadoc
Declaration
Search
Console



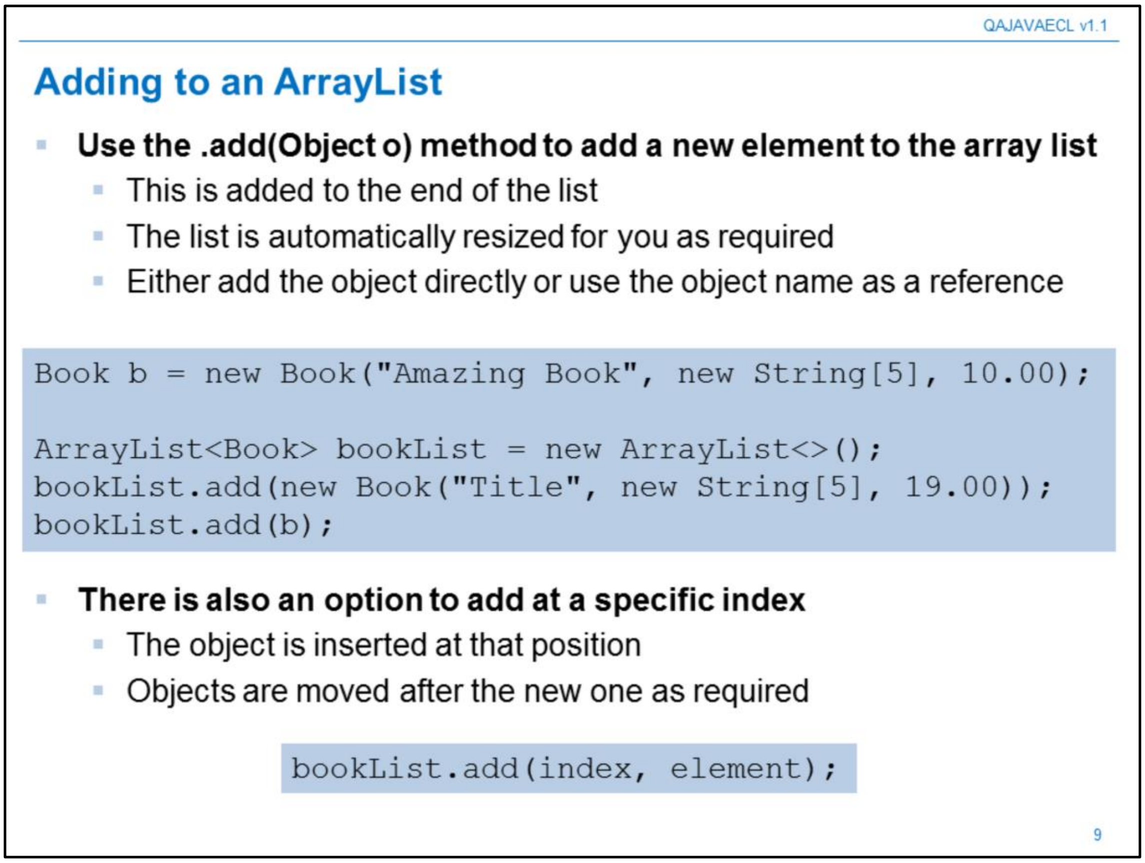
QAJAVAECL v1.1

Lists and ArrayList

- **A list is an ordered collection**
  - Also known as a sequence
  - Stores things one after another in the order they were added
- **ArrayLists can be of any size**
  - It does not need to be specified up front
  - The type of the ArrayList needs to be specified. This is done in the angular brackets <>
  - If we are using Java 7 and beyond we do not need to specify the object type after the new statement

```
ArrayList<Object> arrayList = new ArrayList<Object>();
//java 7+ only
ArrayList<Object> arrayList = new ArrayList<>();
```

The angular brackets mean that this is a generic class, so the object type used by the class is given when it is created.



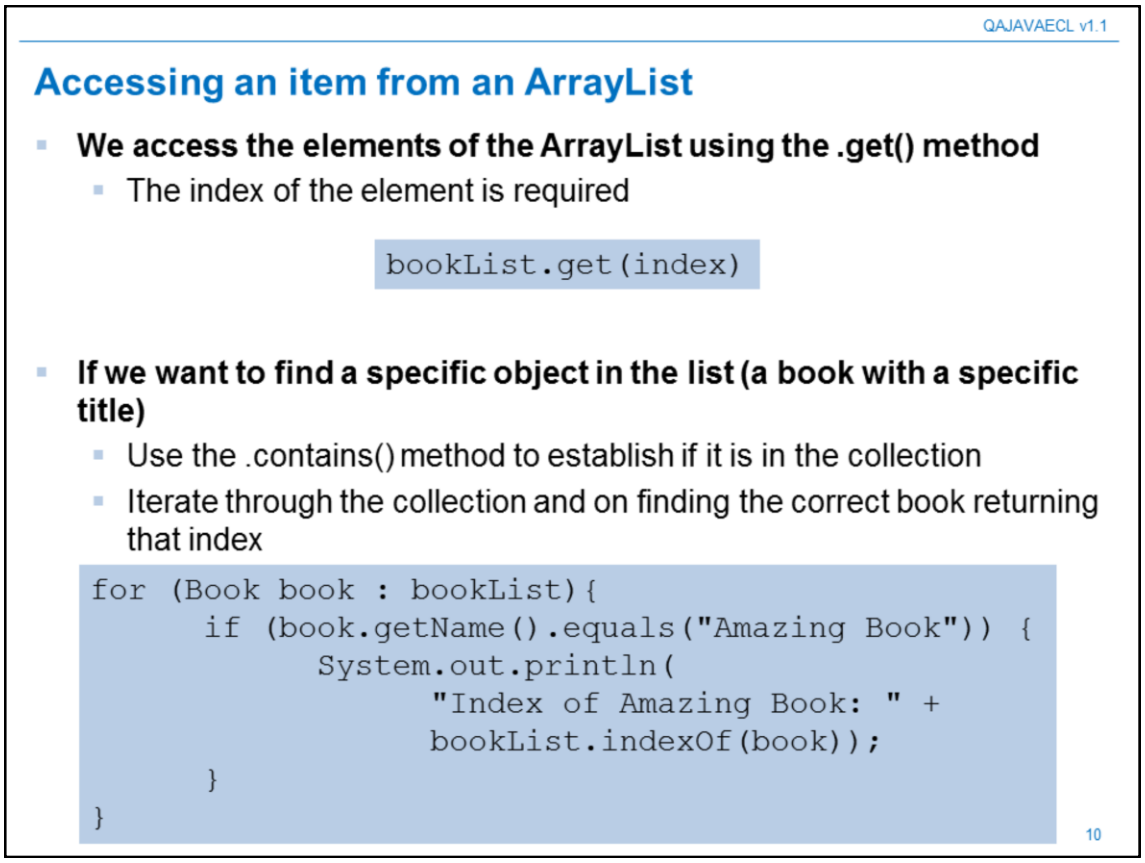
Adding to an ArrayList

- **Use the .add(Object o) method to add a new element to the array list**
  - This is added to the end of the list
  - The list is automatically resized for you as required
  - Either add the object directly or use the object name as a reference

```
Book b = new Book("Amazing Book", new String[5], 10.00);

ArrayList<Book> bookList = new ArrayList<>();
bookList.add(new Book("Title", new String[5], 19.00));
bookList.add(b);
```

- **There is also an option to add at a specific index**
  - The object is inserted at that position
  - Objects are moved after the new one as required

```
bookList.add(index, element);
```



QAJAVAECL v1.1

Accessing an item from an ArrayList

- **We access the elements of the ArrayList using the .get() method**
  - The index of the element is required

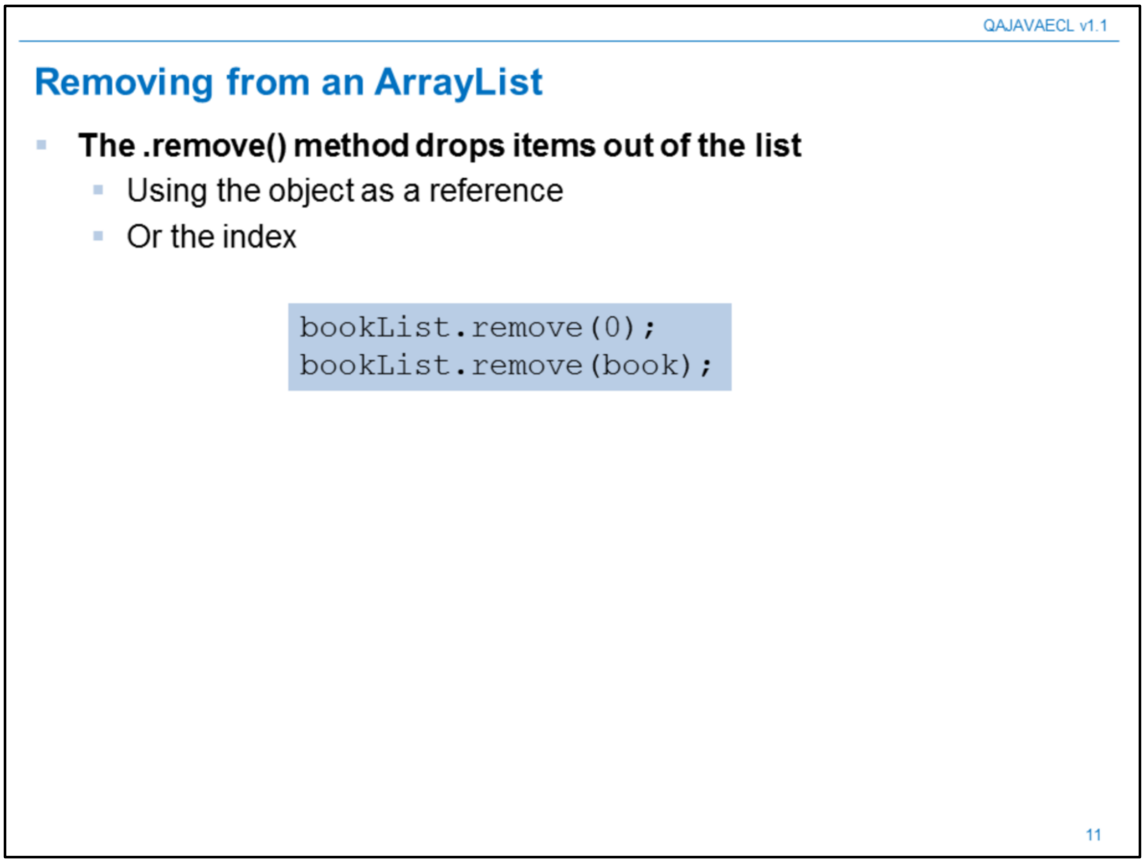
```
bookList.get(index)
```

- **If we want to find a specific object in the list (a book with a specific title)**
  - Use the .contains() method to establish if it is in the collection
  - Iterate through the collection and on finding the correct book returning that index

```
for (Book book : bookList){
      if (book.getName().equals("Amazing Book")) {
            System.out.println(
                  "Index of Amazing Book: " +
                  bookList.indexOf(book));
      }
}
```

10

If we used a standard for loop with a loop counter then we could just print the value of the current counter. In this case we use indexOf to get the index of the object that matches the one we have found. (This may not be the most efficient way of printing the index!)



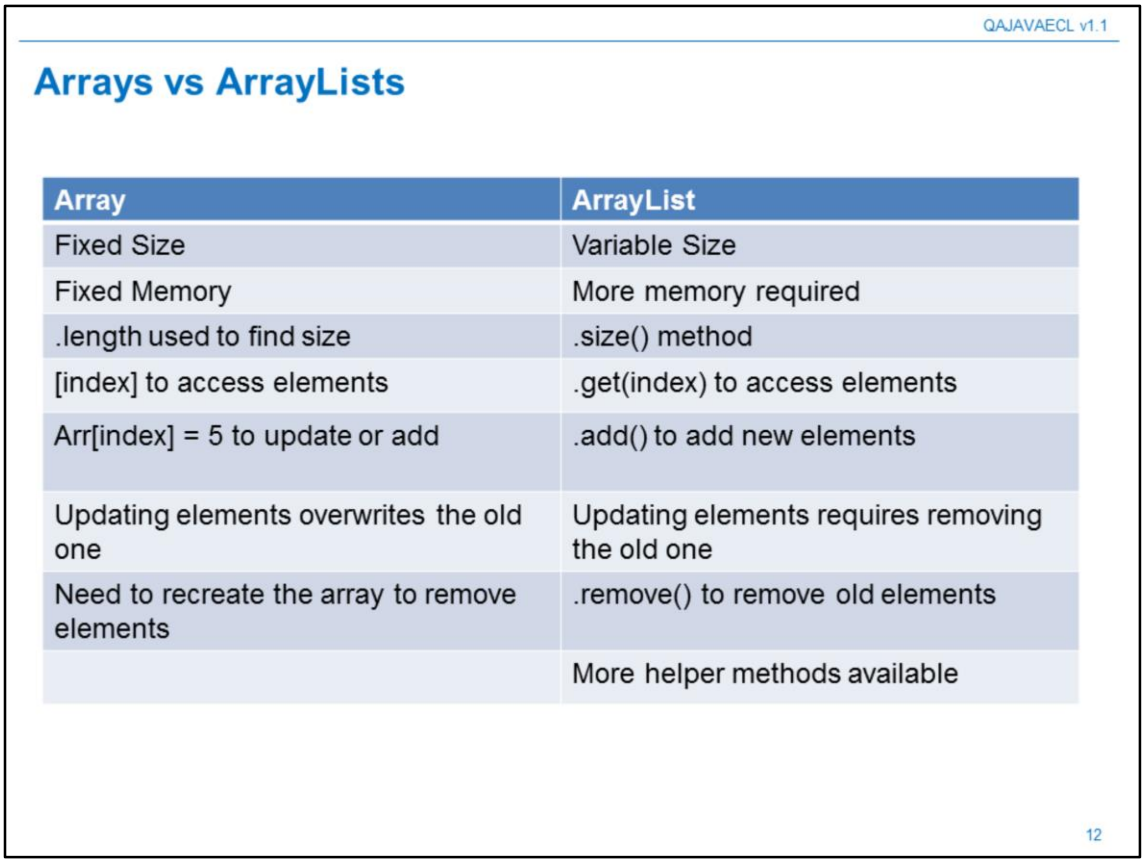
QAJAVAECL v1.1

Removing from an ArrayList

- **The .remove() method drops items out of the list**
  - Using the object as a reference
  - Or the index

```
bookList.remove(0);
bookList.remove(book);
```

11

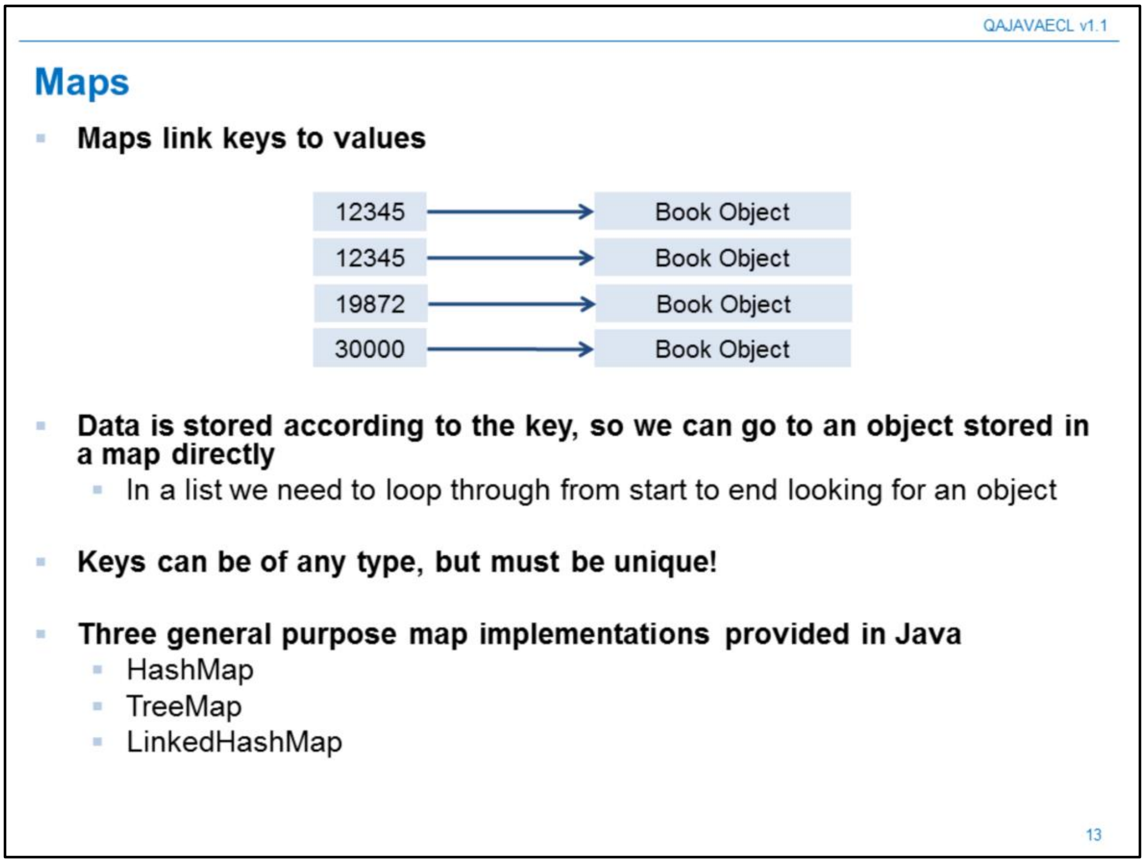


QAJAVAECL v1.1

Arrays vs ArrayLists

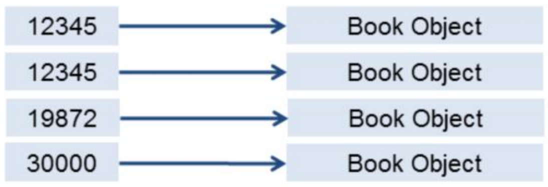
Array	ArrayList
Fixed Size	Variable Size
Fixed Memory	More memory required
.length used to find size	.size() method
[index] to access elements	.get(index) to access elements
Arr[index] = 5 to update or add	.add() to add new elements
Updating elements overwrites the old one	Updating elements requires removing the old one
Need to recreate the array to remove elements	.remove() to remove old elements
	More helper methods available

12

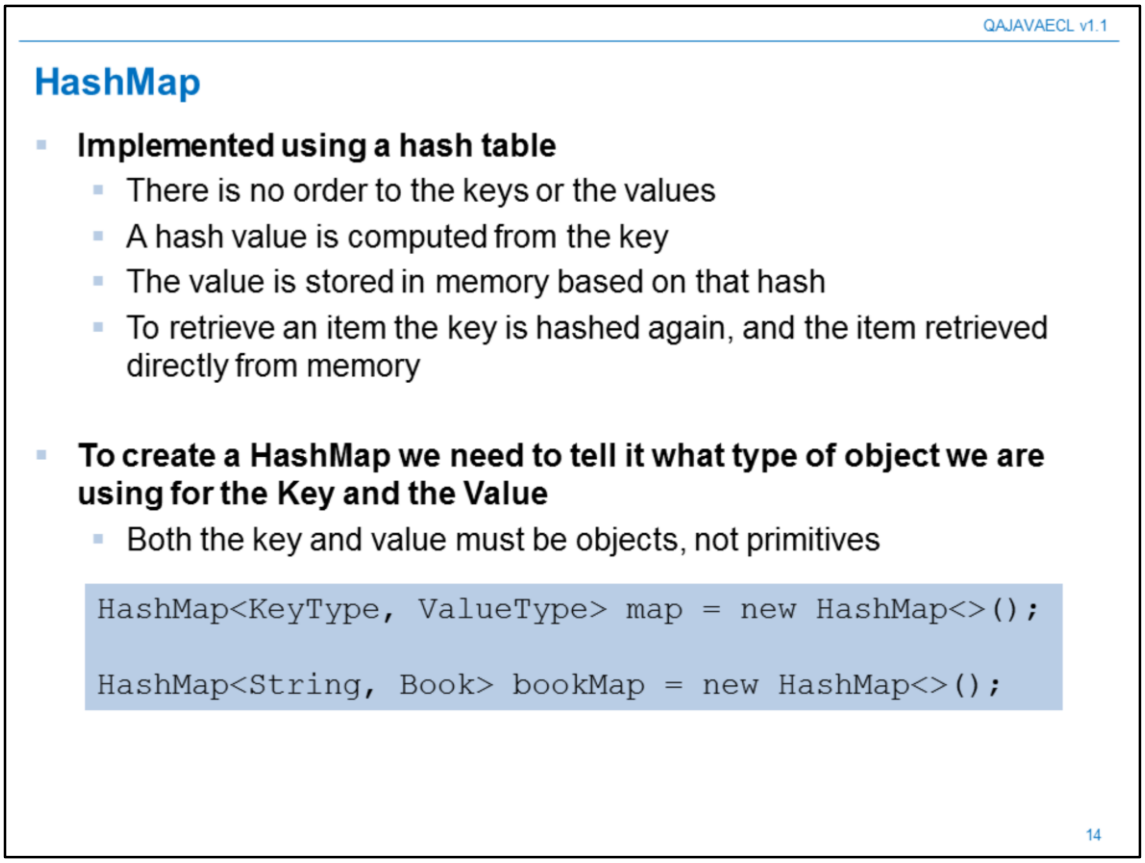


Maps

- **Maps link keys to values**



- **Data is stored according to the key, so we can go to an object stored in a map directly**
  - In a list we need to loop through from start to end looking for an object
- **Keys can be of any type, but must be unique!**
- **Three general purpose map implementations provided in Java**
  - HashMap
  - TreeMap
  - LinkedHashMap

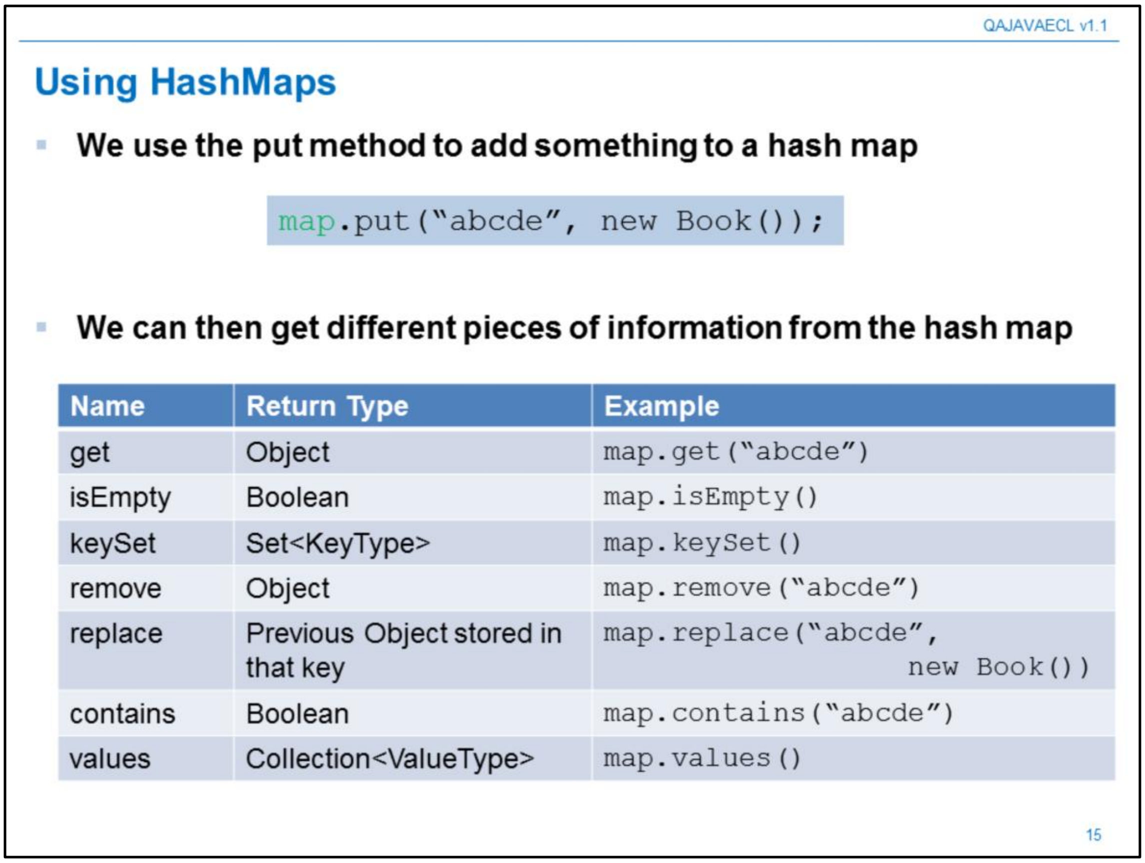


HashMap

- **Implemented using a hash table**
  - There is no order to the keys or the values
  - A hash value is computed from the key
  - The value is stored in memory based on that hash
  - To retrieve an item the key is hashed again, and the item retrieved directly from memory

- **To create a HashMap we need to tell it what type of object we are using for the Key and the Value**
  - Both the key and value must be objects, not primitives

```
HashMap<KeyType, ValueType> map = new HashMap<>();

HashMap<String, Book> bookMap = new HashMap<>();
```



QAJAVAECL v1.1

Using HashMaps

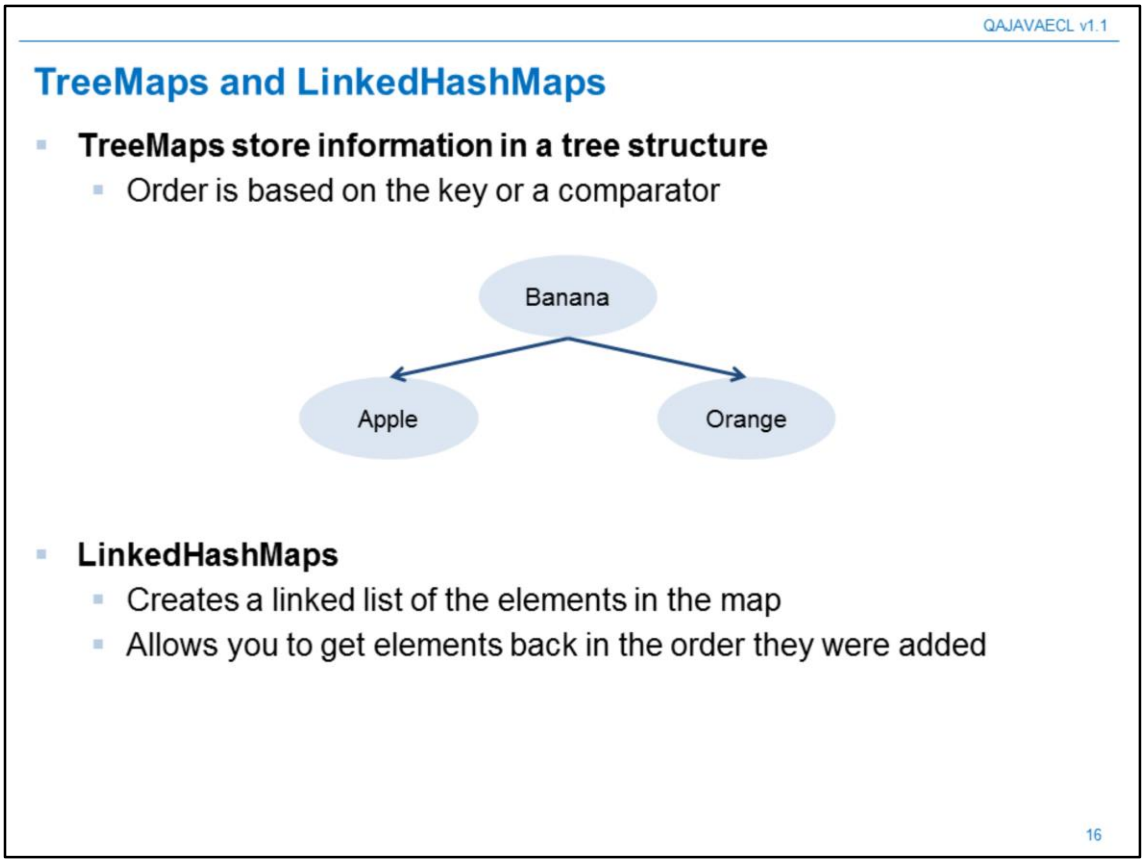
- We use the put method to add something to a hash map

```
map.put("abcde", new Book());
```

- We can then get different pieces of information from the hash map

Name	Return Type	Example
get	Object	`map.get("abcde")`
isEmpty	Boolean	`map.isEmpty()`
keySet	Set<KeyType>	`map.keySet()`
remove	Object	`map.remove("abcde")`
replace	Previous Object stored in that key	`map.replace("abcde", new Book())`
contains	Boolean	`map.contains("abcde")`
values	Collection<ValueType>	`map.values()`

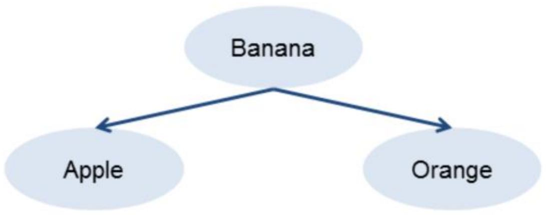
15



QAJAVAECL v1.1

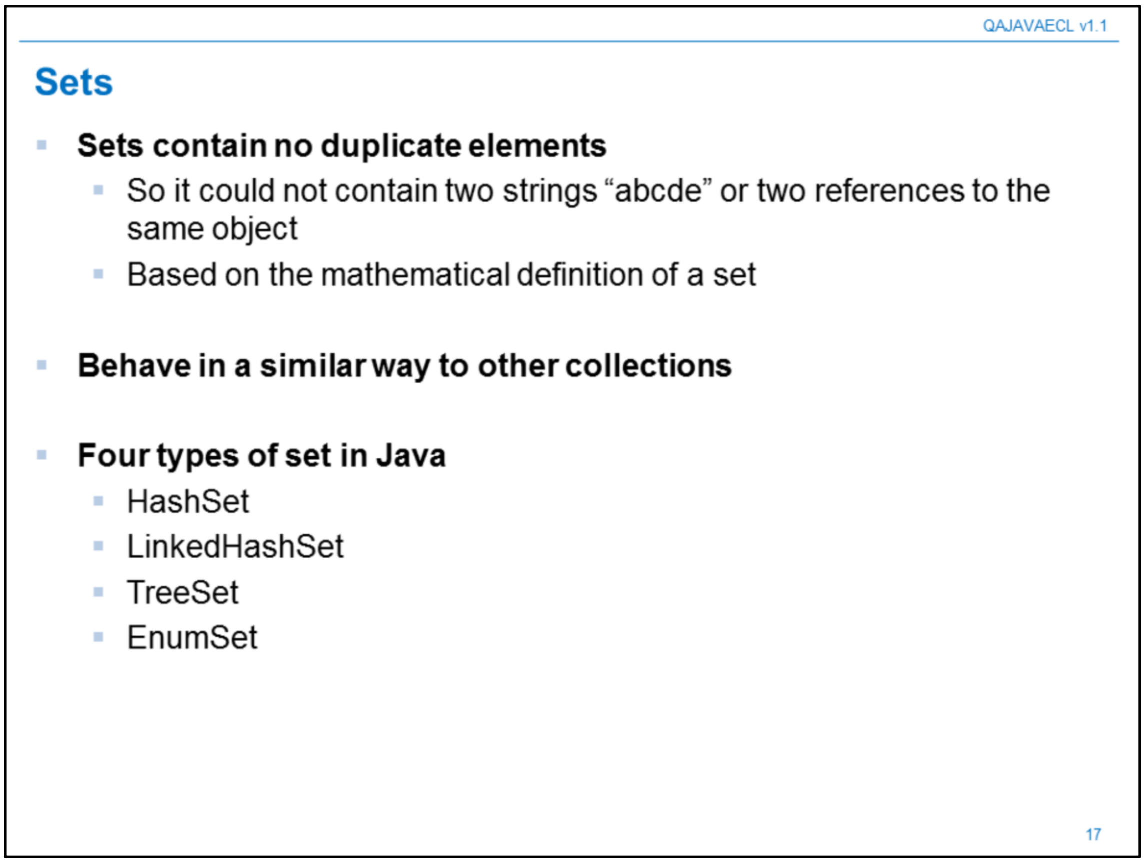
TreeMaps and LinkedHashMaps

- **TreeMaps store information in a tree structure**
  - Order is based on the key or a comparator



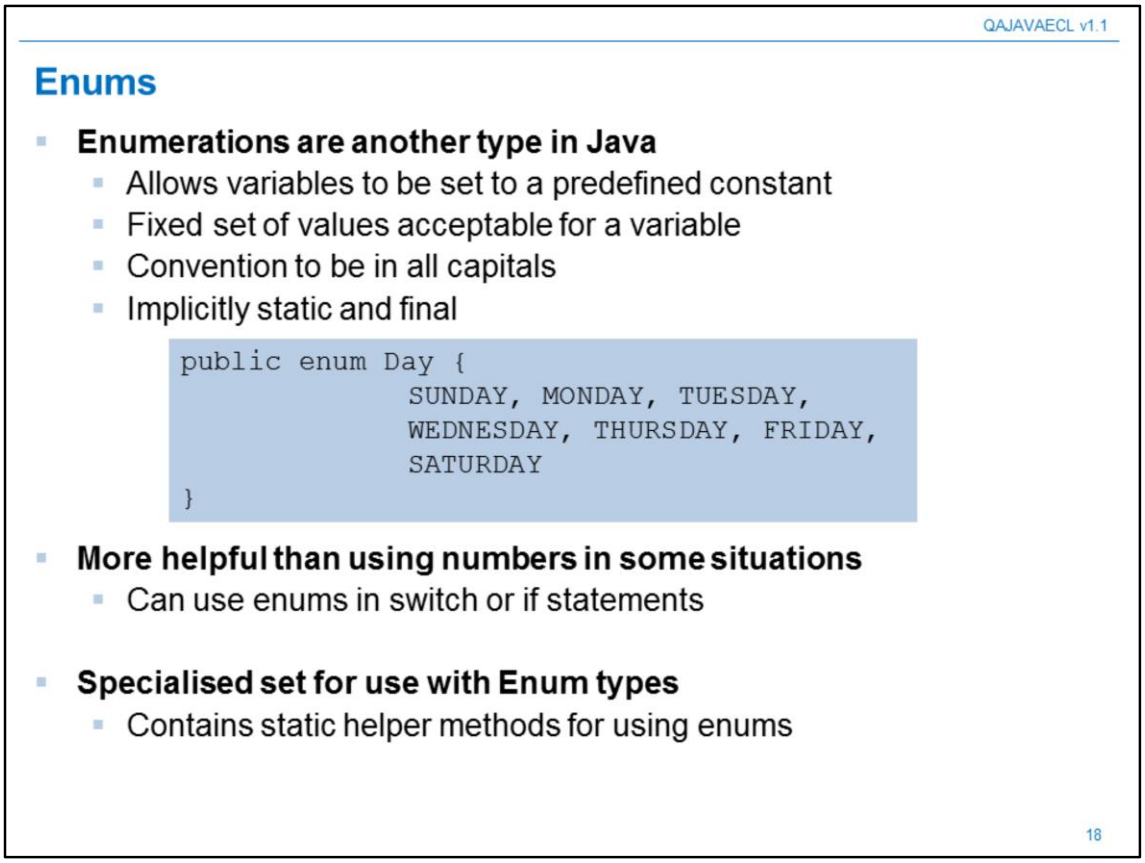
- **LinkedHashMaps**
  - Creates a linked list of the elements in the map
  - Allows you to get elements back in the order they were added

16



Sets

- **Sets contain no duplicate elements**
  - So it could not contain two strings “abcde” or two references to the same object
  - Based on the mathematical definition of a set
- **Behave in a similar way to other collections**
- **Four types of set in Java**
  - HashSet
  - LinkedHashSet
  - TreeSet
  - EnumSet



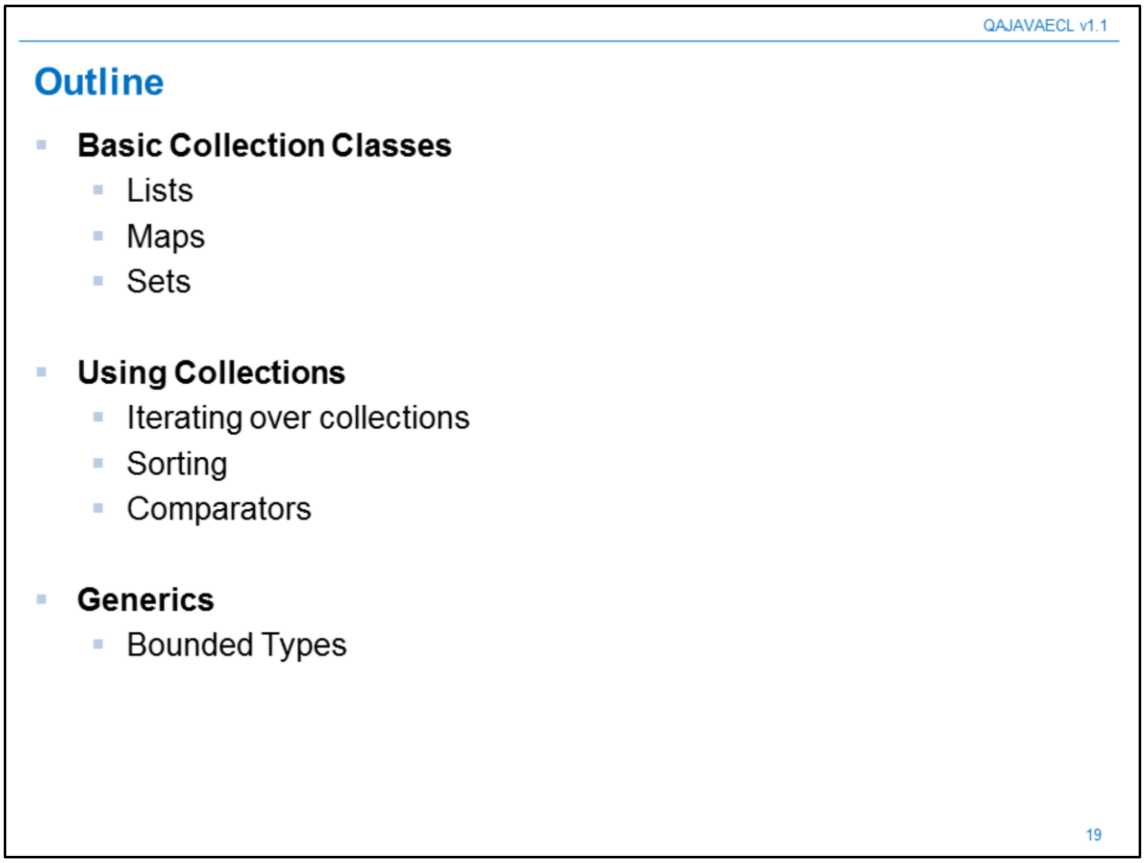
Enums

- **Enumerations are another type in Java**
  - Allows variables to be set to a predefined constant
  - Fixed set of values acceptable for a variable
  - Convention to be in all capitals
  - Implicitly static and final

```
public enum Day {
                SUNDAY, MONDAY, TUESDAY,
                WEDNESDAY, THURSDAY, FRIDAY,
                SATURDAY
}
```

- **More helpful than using numbers in some situations**
  - Can use enums in switch or if statements
- **Specialised set for use with Enum types**
  - Contains static helper methods for using enums

18

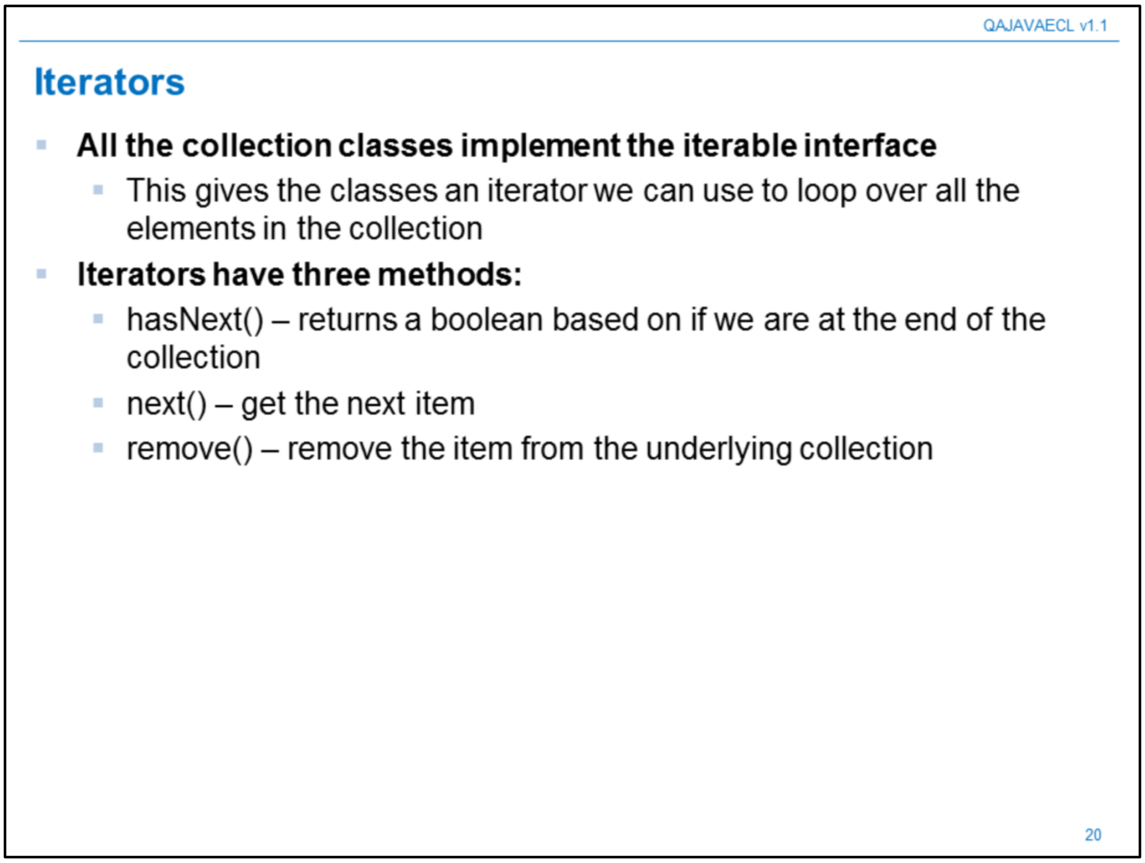


QAJAVAECL v1.1

Outline

- **Basic Collection Classes**
  - Lists
  - Maps
  - Sets
- **Using Collections**
  - Iterating over collections
  - Sorting
  - Comparators
- **Generics**
  - Bounded Types

19

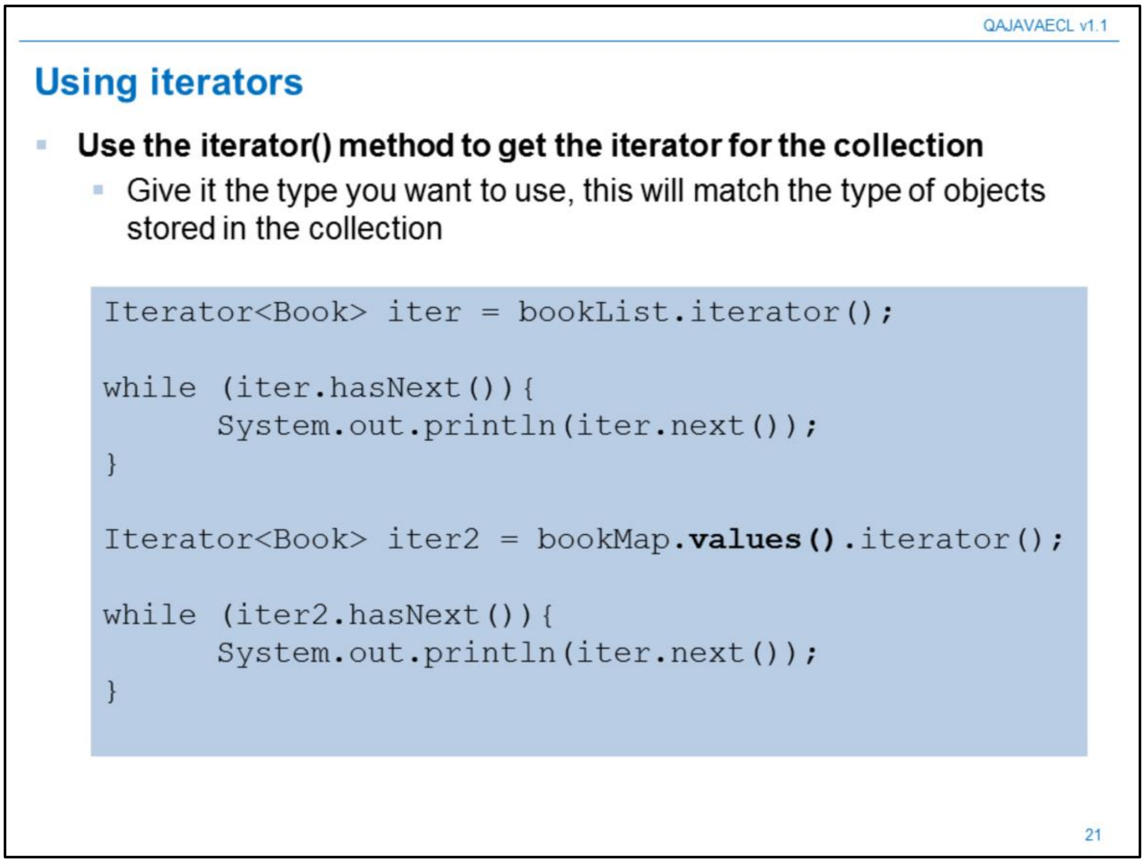


QAJAVAECL v1.1

Iterators

- **All the collection classes implement the iterable interface**
  - This gives the classes an iterator we can use to loop over all the elements in the collection
- **Iterators have three methods:**
  - hasNext() – returns a boolean based on if we are at the end of the collection
  - next() – get the next item
  - remove() – remove the item from the underlying collection

20



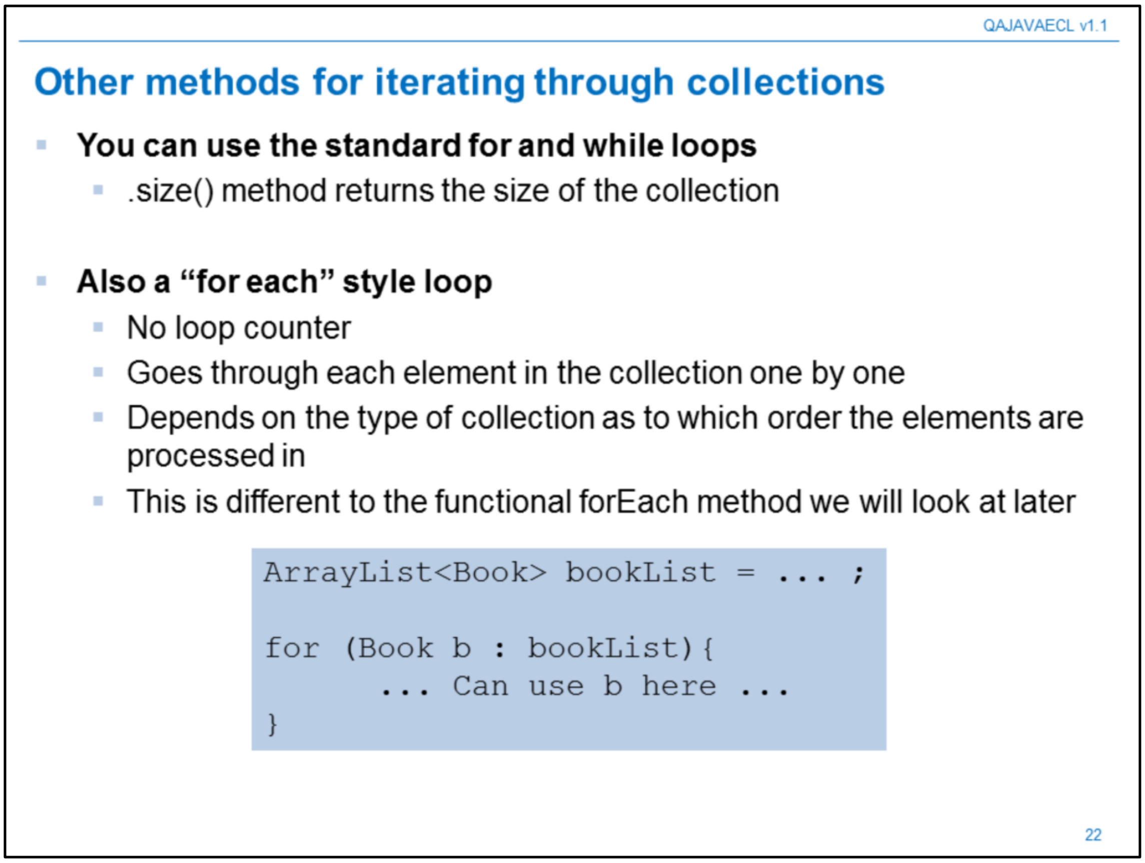
QAJAVAECL v1.1

Using iterators

- **Use the iterator() method to get the iterator for the collection**
  - Give it the type you want to use, this will match the type of objects stored in the collection

```
Iterator<Book> iter = bookList.iterator();

while (iter.hasNext()){
      System.out.println(iter.next());
}

Iterator<Book> iter2 = bookMap.values().iterator();

while (iter2.hasNext()){
      System.out.println(iter.next());
}
```

21



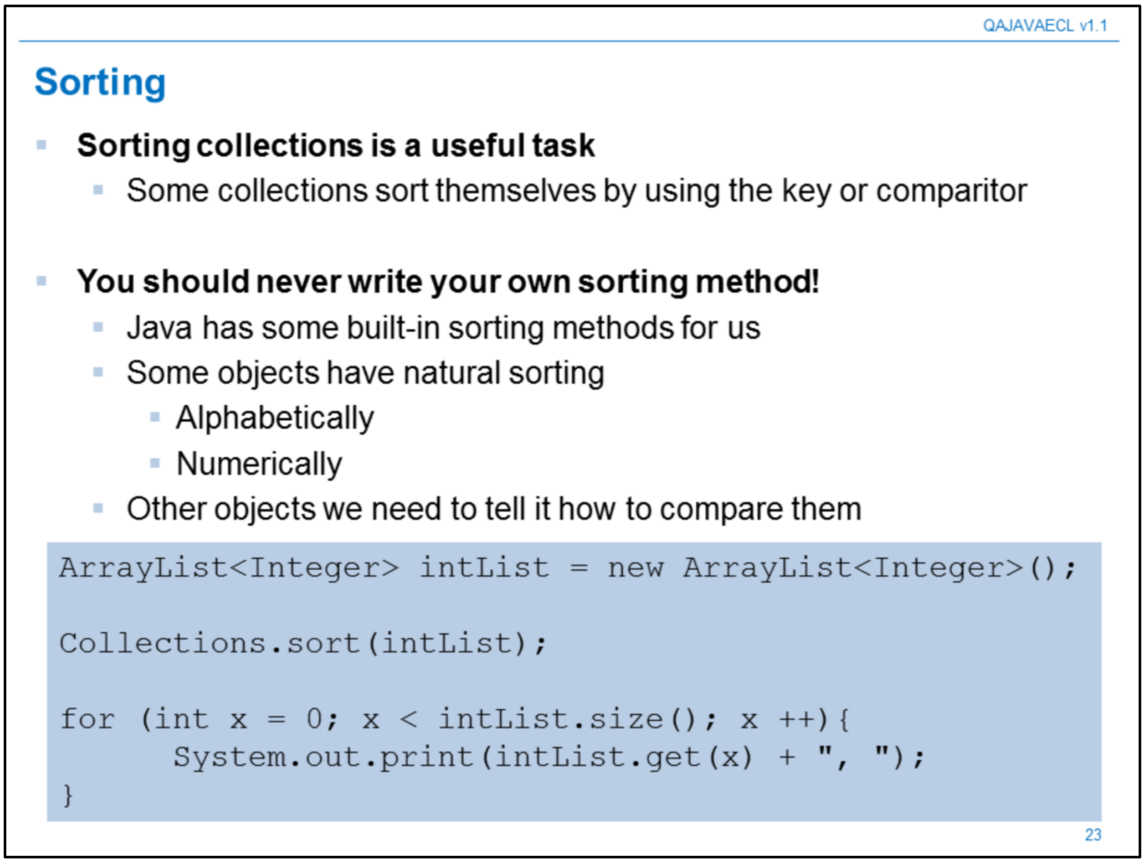
QAJAVAECL v1.1

Other methods for iterating through collections

- **You can use the standard for and while loops**
  - .size() method returns the size of the collection
- **Also a "for each" style loop**
  - No loop counter
  - Goes through each element in the collection one by one
  - Depends on the type of collection as to which order the elements are processed in
  - This is different to the functional forEach method we will look at later

```
ArrayList<Book> bookList = ... ;

for (Book b : bookList){
        ... Can use b here ...
}
```

22



QAJAVAECL v1.1

Sorting

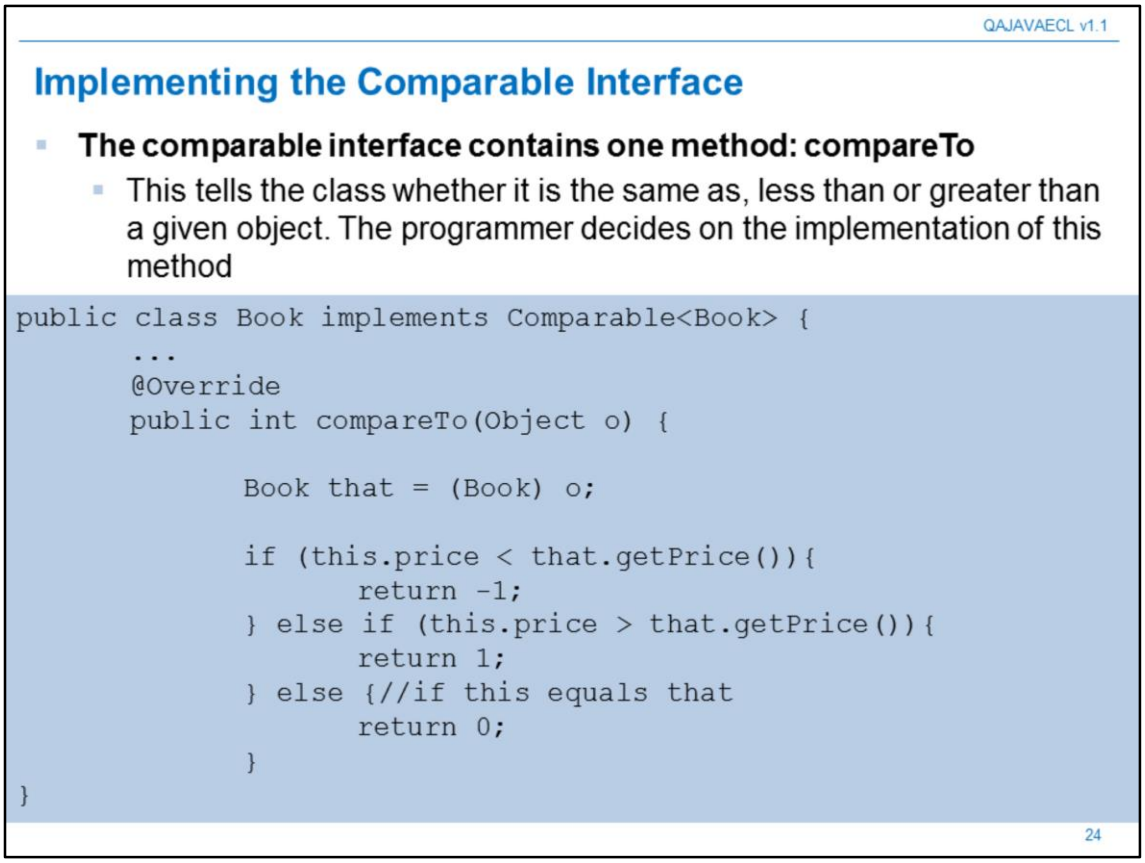
- **Sorting collections is a useful task**
  - Some collections sort themselves by using the key or comparitor
- **You should never write your own sorting method!**
  - Java has some built-in sorting methods for us
  - Some objects have natural sorting
    - Alphabetically
    - Numerically
  - Other objects we need to tell it how to compare them

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

Collections.sort(intList);

for (int x = 0; x < intList.size(); x ++){
      System.out.print(intList.get(x) + ", ");
}
```

23

There is an alternative method in the Arrays class for sorting arrays

```
int[] intArr = {11,53,1,2,6,22,99};

Arrays.sort(intArr);

for (int x = 0; x < intArr.length; x ++){
     System.out.print(intArr[x] + ", ");
}
```



QAJAVAECL v1.1

Implementing the Comparable Interface

- **The comparable interface contains one method: compareTo**
  - This tells the class whether it is the same as, less than or greater than a given object. The programmer decides on the implementation of this method

```
public class Book implements Comparable<Book> {
      ...
      @Override
      public int compareTo(Object o) {

            Book that = (Book) o;

            if (this.price < that.getPrice()){
                  return -1;
            } else if (this.price > that.getPrice()){
                  return 1;
            } else {//if this equals that
                  return 0;
            }
}
```

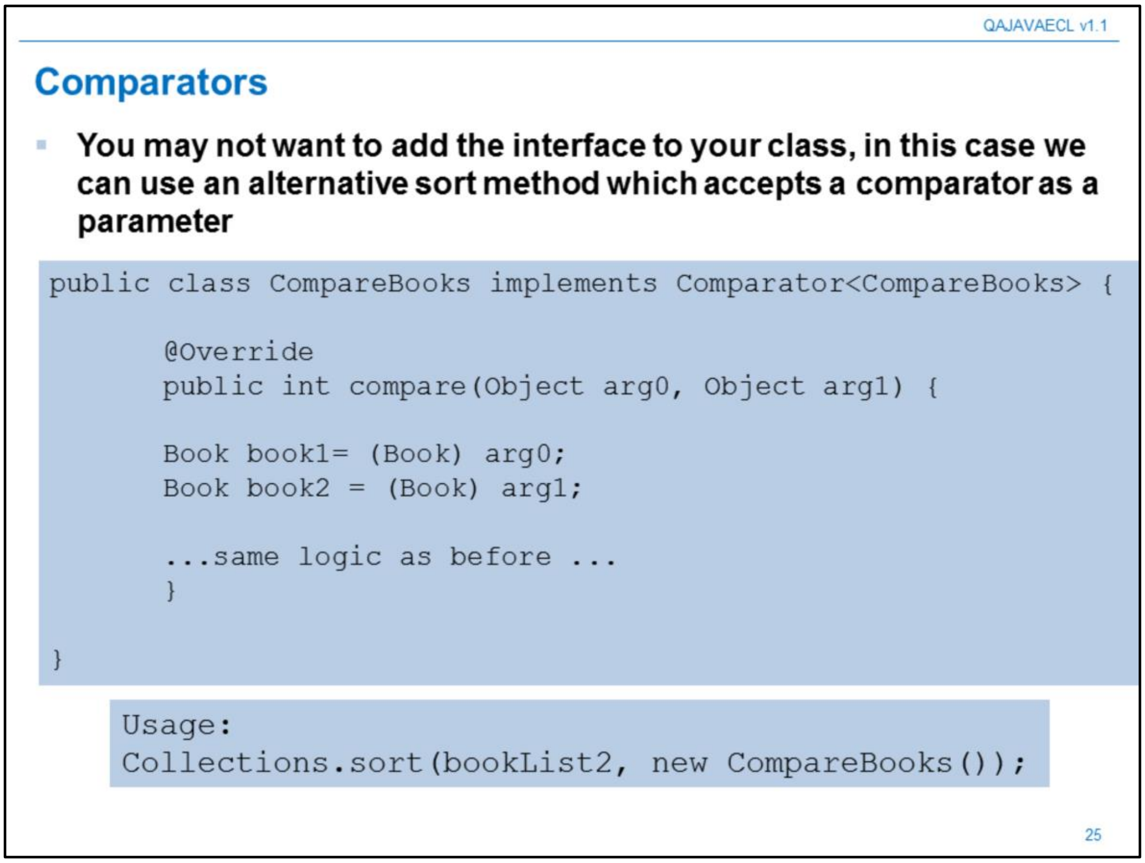
24

This is sorting the books based on their price. If this book's price is less than that book's price it returns -1 (the lower value) if it is higher it returns 1, if they are the same it returns 0.

This class is called by the Collections.sort() method allowing it to sort the collection by the given order

ArrayList<Book> bookList = new ArrayList<>();

Collections.sort(bookList,new Book ());



QAJAVAECL v1.1

Comparators

- You may not want to add the interface to your class, in this case we can use an alternative sort method which accepts a comparator as a parameter

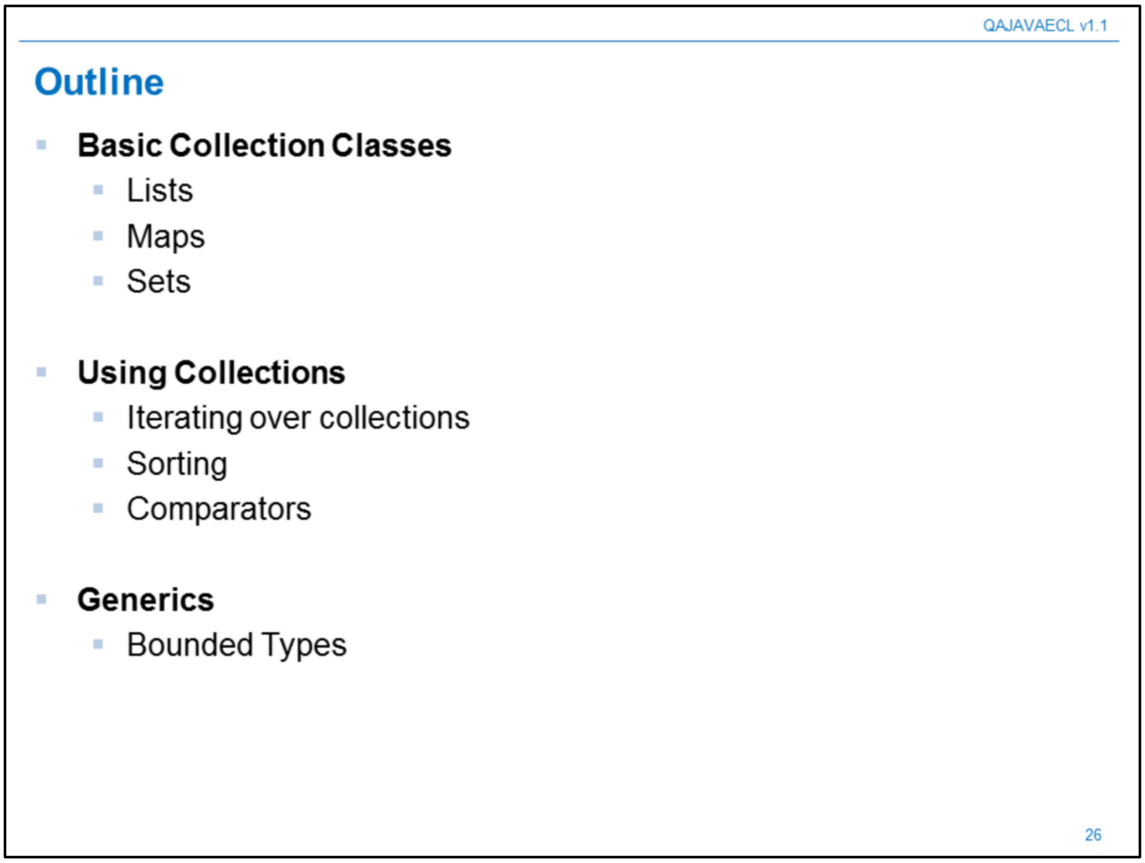
```
public class CompareBooks implements Comparator<CompareBooks> {

        @Override
        public int compare(Object arg0, Object arg1) {

        Book book1= (Book) arg0;
        Book book2 = (Book) arg1;

        ...same logic as before ...
        }

}
```

```
Usage:
Collections.sort(bookList2, new CompareBooks());
```

25

The full code for this class is:

```
public class CompareBooks implements Comparator <CompareBooks> {

@Override
public int compare(Object arg0, Object arg1) {

    Book book1= (Book) arg0;
    Book book2 = (Book) arg1;

    if (book1.getPrice() < book2.getPrice()){
        return -1;
    } else if (book1.getPrice() > book2.getPrice()){
        return 1;
    } else { //if this equals that
        return 0;
    }
}

}
```

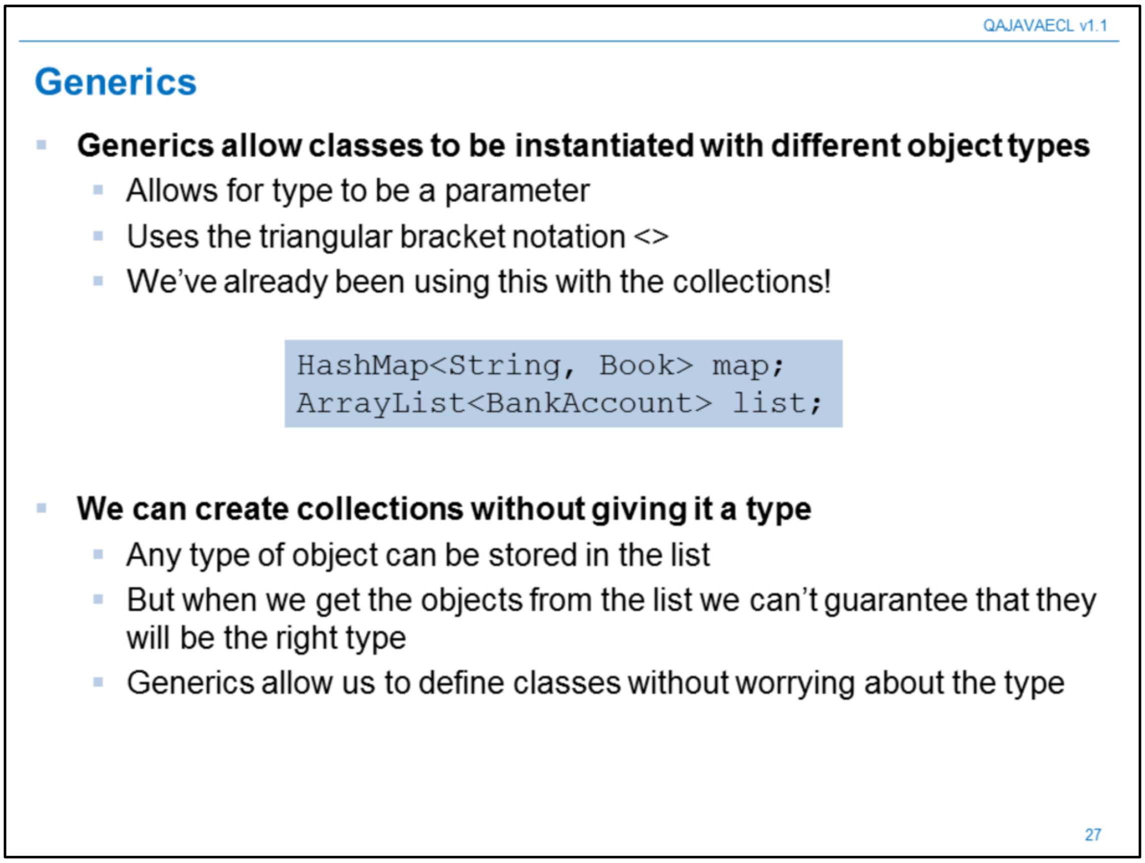


QAJAVAECL v1.1

Outline

- **Basic Collection Classes**
  - Lists
  - Maps
  - Sets
- **Using Collections**
  - Iterating over collections
  - Sorting
  - Comparators
- **Generics**
  - Bounded Types

26



QAJAVAECL v1.1

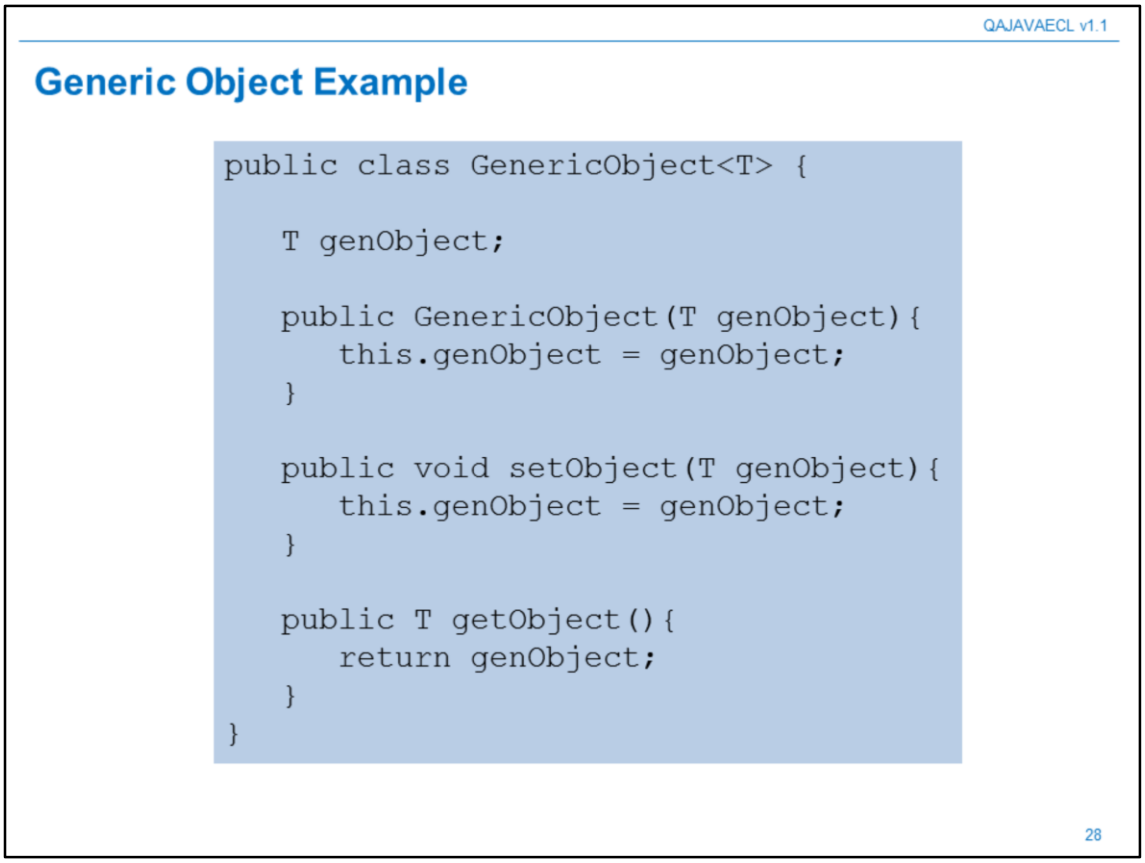
Generics

- **Generics allow classes to be instantiated with different object types**
  - Allows for type to be a parameter
  - Uses the triangular bracket notation <>
  - We've already been using this with the collections!

```
HashMap<String, Book> map;
ArrayList<BankAccount> list;
```

- **We can create collections without giving it a type**
  - Any type of object can be stored in the list
  - But when we get the objects from the list we can't guarantee that they will be the right type
  - Generics allow us to define classes without worrying about the type

27

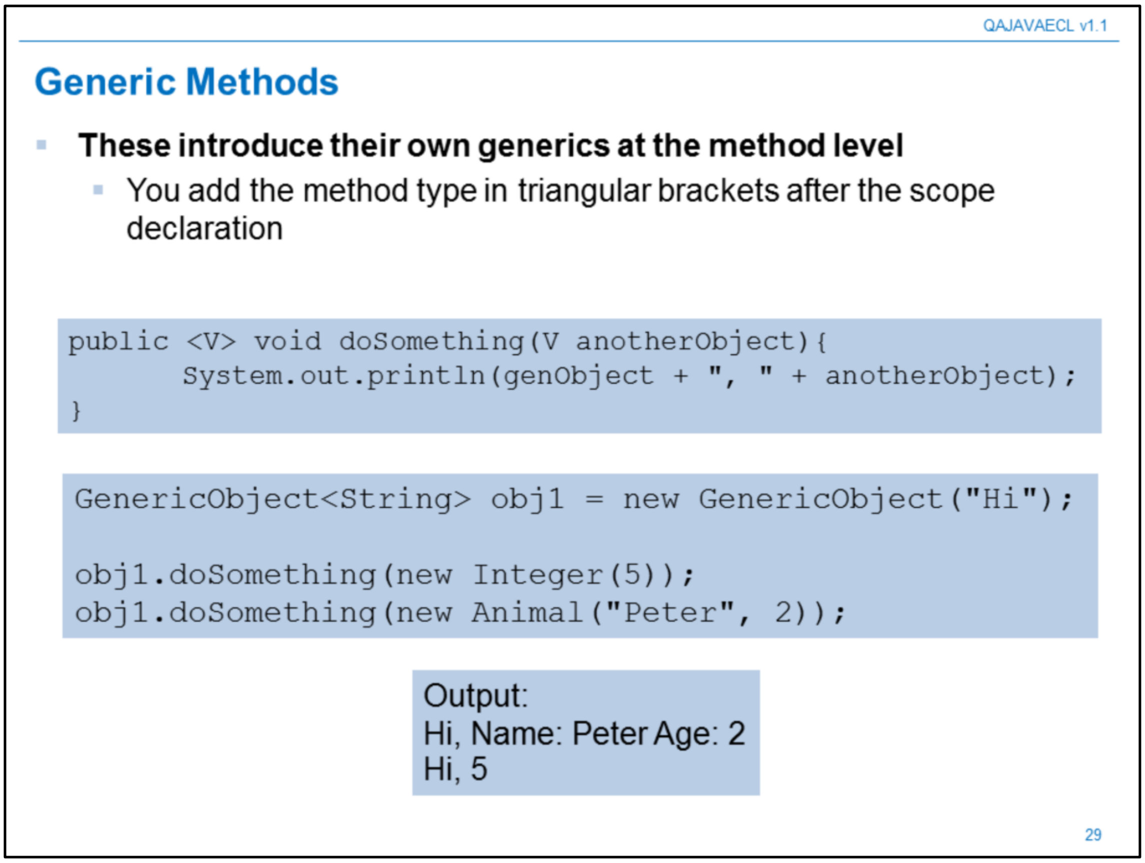


QAJAVAECL v1.1

Generic Object Example

```
public class GenericObject<T> {

   T genObject;

   public GenericObject(T genObject){
      this.genObject = genObject;
   }

   public void setObject(T genObject){
      this.genObject = genObject;
   }

   public T getObject(){
      return genObject;
   }
}
```

28



Generic Methods

- **These introduce their own generics at the method level**
  - You add the method type in triangular brackets after the scope declaration

```
public <V> void doSomething(V anotherObject){
      System.out.println(genObject + ", " + anotherObject);
}
```

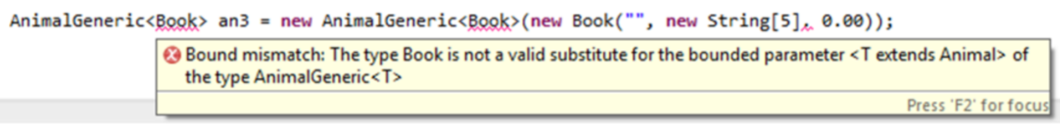
```
GenericObject<String> obj1 = new GenericObject("Hi");

obj1.doSomething(new Integer(5));
obj1.doSomething(new Animal("Peter", 2));
```

```
Output:
Hi, Name: Peter Age: 2
Hi, 5
```

Bounded Types

- **It is also possible to say that the type should exist in some hierarchy**
  - For example: Allow any type in this class as long as it is a sub-type of Animal

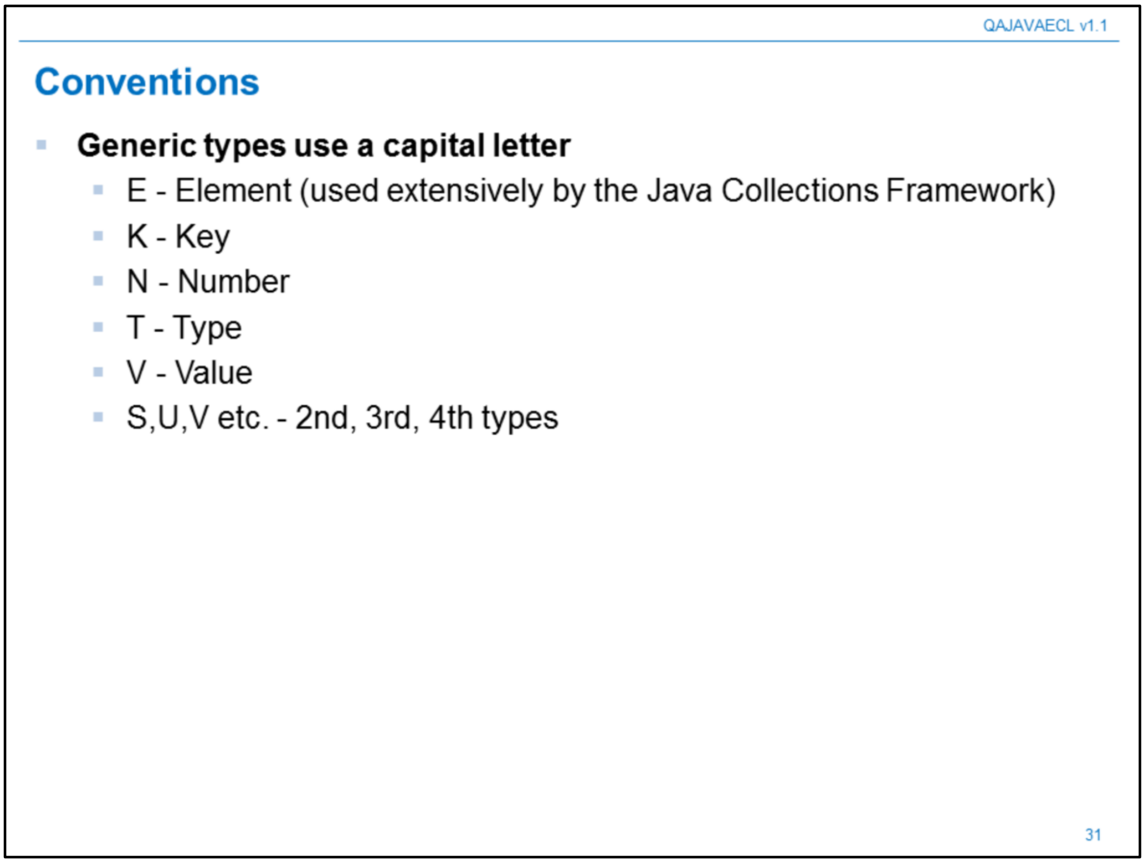
```
public class AnimalGeneric<T extends Animal> {
      ...
}

AnimalGeneric<Rabbit> an1 = new AnimalGeneric<Rabbit>
                                (new Rabbit("Peter", 2, true));
AnimalGeneric<Animal> an2 = new AnimalGeneric<Animal>
                                (new Rabbit("Benjy", 3, true));
```



```
AnimalGeneric<Book> an3 = new AnimalGeneric<Book>(new Book("", new String[5], 0.00));
```

Bound mismatch: The type Book is not a valid substitute for the bounded parameter <T extends Animal> of the type AnimalGeneric<T>

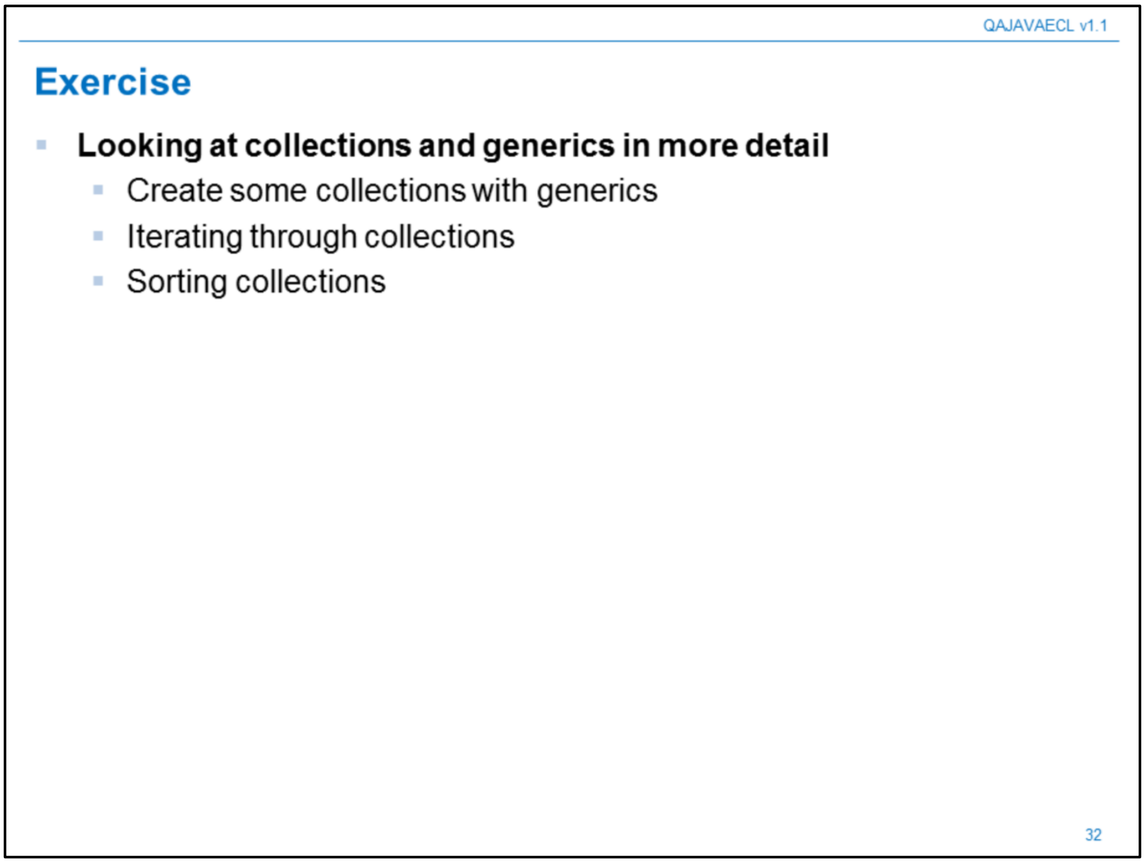
Press 'F2' for focus



Conventions

- **Generic types use a capital letter**
  - E - Element (used extensively by the Java Collections Framework)
  - K - Key
  - N - Number
  - T - Type
  - V - Value
  - S,U,V etc. - 2nd, 3rd, 4th types

More details can be found at:
http://docs.oracle.com/javase/tutorial/java/generics/types.html

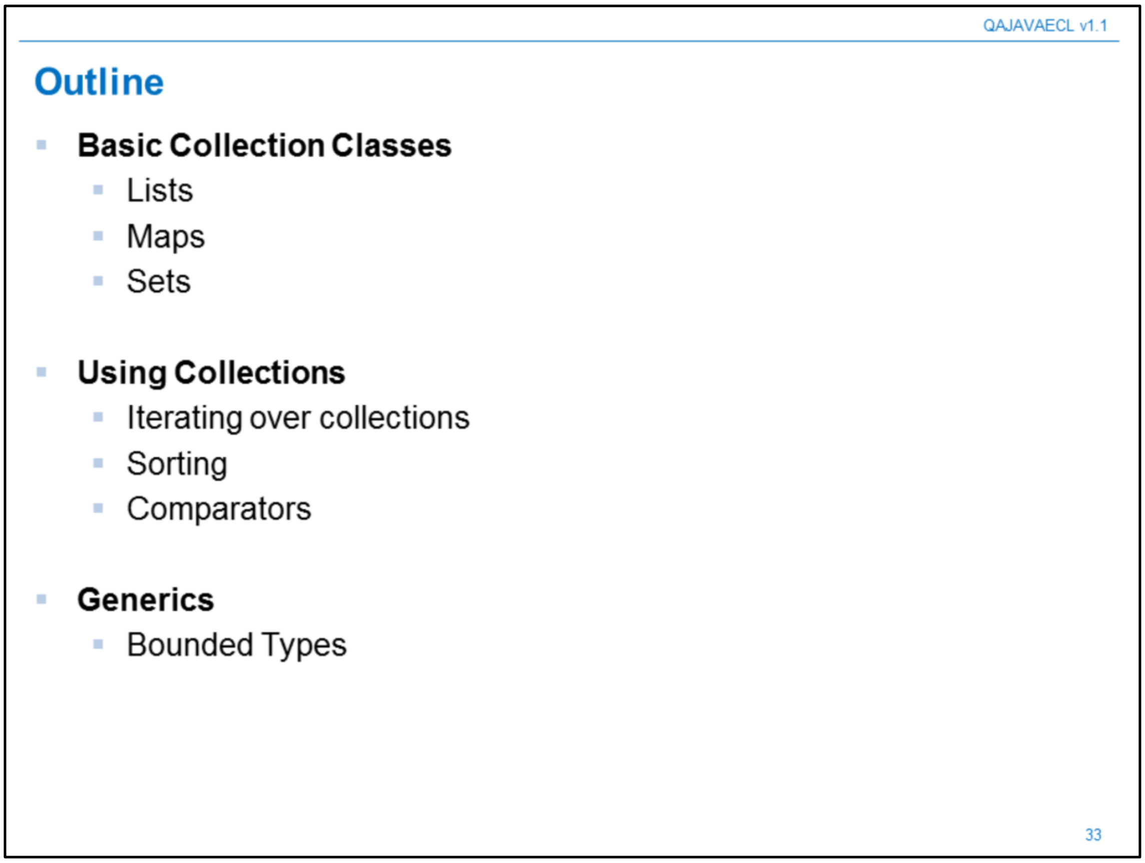


QAJAVAECL v1.1

Exercise

- **Looking at collections and generics in more detail**
  - Create some collections with generics
  - Iterating through collections
  - Sorting collections

32



Outline

- **Basic Collection Classes**
  - Lists
  - Maps
  - Sets
- **Using Collections**
  - Iterating over collections
  - Sorting
  - Comparators
- **Generics**
  - Bounded Types