
ORACLE
Academy

Java Programming

3-2

Collections – Part 1

ORACLE
Academy

Overview

- This lesson covers the following topics:
  - Create a collection without using generics
  - Create a collection using generics
  - Implement an ArrayList
  - Implement a Set

Collections

- We have seen previously that we can use arrays to store items of the same type
- The main issue with arrays is that they have to be manually sized before use
- Arrays are often used to store primitive data up to a pre-set quantity

A collection called ArrayList has already been introduced, this will be explored in more detail along with other collections available in Java.

Collections Without Generics Example

1. Create a project named cells
2. Create this Cell class:

```
public class Cell {
   private String data;

   public void setValue(String celldata) {
      data = celldata;
   }//end method setValue

   public String getValue() {
      return data;
   }//end method get

   public String toString() {
      return data;
   }//end method toString
}//end class Cell
```

- This is a non generic Cell class

Collections Without Generics Example

3. Create a class that will act as your own simple data structure (collection) to store multiple Cell occurrences

- In its current state this is not that useful
- We would have to create more methods, such as insert, sort and delete

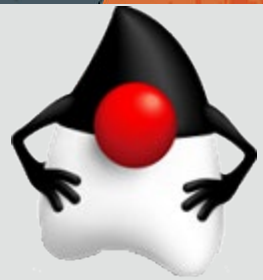
```
public class CellCollection {
  Cell[] cells;
  int index=0;

  public CellCollection(int size) {
    cells = new Cell[size];
  }//end constructor

  public void add(Cell c) {
    cells[index]=c;
    index++;
  }//end method add

  public Cell get(int i) {
    return cells[i];
  }//end method get
    //more methods…..
}//end class CellCollection
```

Collections Without Generics Task



- **Quick Task**: Understanding code

a) What is the purpose of the instance fields?

b) What does the constructor do?

c) What does the add method do?

d) What does the get method do?

Collections Without Generics Suggested Solution

- **Quick Task:** Updating the constructor calls

a) Instance fields

Cell[] cells – Allows the collection to hold an array of Cell objects

int index – Holds the position of the next empty element in the cells array

b) Constructor

```
public CellCollection(int size) {
        cells = new Cell[size];
}//end constructor
```

– Creates an array of cell objects that has a size based on the given parameter

Collections Without Generics Suggested Solution

- **Quick Task**: Updating the constructor calls

c) add method

```
public void add(Cell c) {
        cells[index]=c;
        index++;
}//end method add
```

- Accepts a cell object as a parameter, adds it to the cells array at the position identified by the index field
- Adds 1 to the index to get it ready for the next add operation

Collections Without Generics Suggested Solution

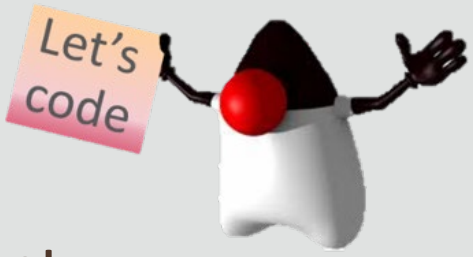
- **Quick Task**: Updating the constructor calls

d) get method

```
public Cell get(int i) {
        return cells[i];
}//end method get
```

- Accepts an integer value that is used to identify the cell object at that position in the cells array
- Returns the cell at that position to give access to that particular cell's properties

Implementing your collection



4. Create the following driver class to test the collection:

```
public class CellCollectionDriver {
   public static void main(String[] args) {
      CellCollection cells = new CellCollection(5);

      cells.add(new Cell());
      cells.add(new Cell());

      System.out.println(cells.get(0));
      System.out.println(cells.get(1));

      cells.get(0).setValue("First Cell");
      cells.get(1).setValue("Second Cell");

      System.out.println(cells.get(0));
      System.out.println(cells.get(1));

   }//end method main
}//end class
```

Implementing your collection explanation

- Create a new array of cells passing the array size as a parameter

```
CellCollection cells = new CellCollection(5);
```

- Add a new cell object to the cells collection using the add method that requires a cell object to be passed

```
cells.add(new Cell());
```

- Display the content of the first two cell objects in the collection

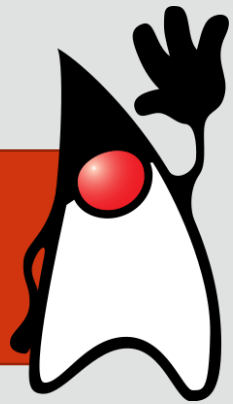
```
System.out.println(cells.get(0));
System.out.println(cells.get(1));
```

Implementing your collection explanation

- Use the get method to return the cell object and then use the setValue method of the Cell class to assign a value to the cell

```
cells.get(0).setValue("First Cell");
cells.get(1).setValue("Second Cell");
```

Implementing your own data structures (collections) is a very complex process and would take a lot of coding to create a fully operational data storage and retrieval system.



Collections With Generics Example

- We could try to create a generic collection similar to our non generic example

```
public class CellGenericCollection<T> {
  T[] cells;
  int index=0;

  public CellGenericCollection(int size) {
    cells = new T[size];
  }//end constructor

  public void add(T c) {
    cells[index]=c;
    index++;
  }//end method add

  public T get(int i) {
    return cells[i];
  }//end method get
}//end class CellCollection
```

Remember that the diamonds around the T - <T> mean that the type will be chosen when an instance is created.

Collections With Generics Example

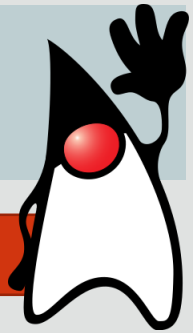
- The problem is we should not create generic arrays in java and this would produce an error:

```
public CellGenericCollection(int size) {
   cells = new T[size];
}//end constructor
```

- The constructor could be modified but this would produce a non type safe solution, therefore removing one of the advantages of generics

```
public CellGenericCollection(int size) {
  cells = (T[]) new Object[size];
}//end constructor
```

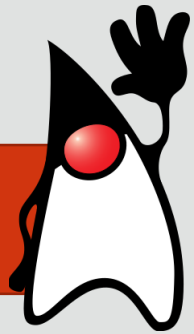
Java has better ways of handling collections of data.



Collections - Introduction

- Data structures are used a lot in programming and as such Java comes with pre-built collection classes for you to utilize
- These save you and every other developer the task of having to create similar storage structures
- They come in a few guises, each with its own pros and cons

Always check if there is a collection that meets your requirements before creating your own to do the exact same task.



Collections

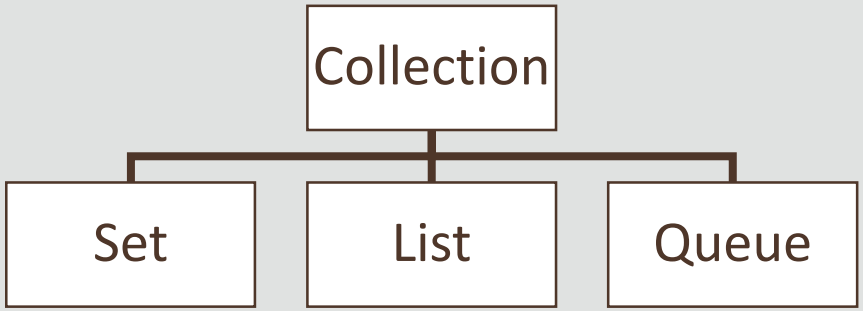
- Collection is an interface in the java.util package that is used to define a group, or collection of objects
- It includes sets and lists and is important in data storage
- Because it is in the java.util package, it will be necessary to import the java.util package into any programs you write using a collection
- This can be done by typing the following class declaration at the top of the program:

```
import java.util.*;
public class ClassExample {
```

If you are using Eclipse and forget to import, then you will be offered the option of the import being added for you.

Collections Class Hierarchy

- Java defines three main interface collections
- This forces any classes that use the collection interface to define its methods such as add()



The Collection interface in Java declares 15 methods that will have to be implemented by any class which uses it.

Lists

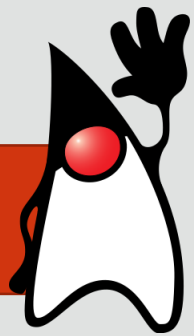
- A list, in Java, is an ordered collection that may contain duplicate elements
- In other words, List extends Collections
- Important qualities of Lists are:
  - They grow and shrink as you add and remove objects
  - They maintain a specific order
  - They allow duplicate elements

List is an interface, so to use lists in java we have to use one of the concrete classes that implements it.

ArrayLists

- Until now we have been using arrays inside a collections class to create a collections interface
- With arrays, you are restricted to the size of the array that you initiate inside the constructor
- ArrayLists are very similar to arrays, except that you do not need to know the size of the ArrayList when you initialize it, like you would with an array

ArrayList is one of the concrete classes that implements the List interface.



Code to Initialize an ArrayList

- You may alter the size of the ArrayList by adding or removing elements
- The only information you need when initializing an ArrayList is the object type that it stores
- The following code initializes an ArrayList of Accounts

```
ArrayList<Account> account= new ArrayList<Account>();
```

Here we make use of the fact that ArrayList supports generics.
The above could be simplified as:
ArrayList<Account> account= new ArrayList();

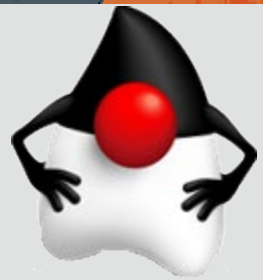
ArrayList Methods

- The ArrayList class already exists in Java. It contains many methods, including the following:

Method	Method Description
`boolean add(E e)`	Appends the specified element to the end of the list.
`void add(int index, E element)`	Inserts the specified element at the specified position
`E get(int index)`	Returns the element at the specified position.
`E set(int index, E element)`	Replaces the element at the specified position in the list with the specified element.
`E remove(int index)`	Removes the element at the specified position in the list
`boolean remove(Object o)`	Removes the first occurrence of the specified element from this list, if it is present.

More ArrayList methods can be found in the Java API

ArrayList Task



a) Create a project named classlist
b) Create a class Student that has a main method
c) Initialize an ArrayList of Strings called studentNames in the main method
d) Create a static method named addStudents that adds each student in your classroom to the ArrayList
e) Call the addStudents method from main

ArrayList Task Suggested Solution

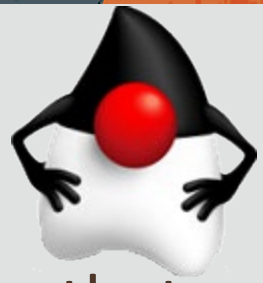
```
import java.util.ArrayList;
public class Student {
   public static void main(String[] args) {
      ArrayList<String> studentNames = new ArrayList();

      addStudents(studentNames);
   }//end method main

   static void addStudents(ArrayList<String> studentNames) {
      //Add the names of everyone in your class
      studentNames.add("Mark");
      studentNames.add("Andrew");
      studentNames.add("Beth");
         .
         .
         .
   }//end method addStudents

}//end class Student
```

ArrayList Task continued



f) Create a static method named displayStudents that uses an enhanced for loop to display the contents of the ArrayList. The output should look like:

```
Student Name: Mark
```

g) Call the displayStudents method at the end of main

h) Collections contains a method called sort that takes in a List as a parameter and lexicographically sorts the list, an ArrayList is a List so use the Collections sort method on your ArrayList at the bottom of main

i) Call the displayStudents method again from main

ArrayList Task Suggested Solution

```
import java.util.ArrayList;
import java.util.Collections;
public class Student {
   public static void main(String[] args) {
      ArrayList<String> studentNames = new ArrayList();

      addStudents(studentNames);
      displayStudents(studentNames);
      Collections.sort(studentNames);
      displayStudents(studentNames);
   }//end method main

   static void displayStudents(ArrayList<String> studentNames) {
      for(String student: studentNames)
         System.out.println("Student Name: " + student);
      //endfor
   }//end method displayStudents

   static void addStudents(ArrayList<String> studentNames) {
   }//end method addStudents
}//end class Student
```

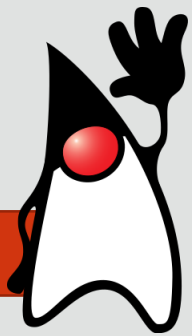
Sets

- A set is a collection of elements that does not contain duplicates
- For example, a set of integers 1, 2, 2, 3, 5, 3, 7 would be:
  - {1, 2, 3, 5, 7}
- All elements of a set must be of the same type
- For example, you can not have a set that includes integers and Strings, but you could have a set of integers and a separate set of Strings
- The Set interface contains only methods inherited from the Collection interface

Implementing Sets with a HashSet

- Lists, which are a collection that may contain duplicates, are implemented through ArrayLists
- Similarly, Sets are commonly implemented with a HashSet
- A HashSet:
  - Is similar to an ArrayList, but does not have any specific ordering
  - Is good to use when order is not important

A HashSet is a concrete class that implements the set interface.



Some HashSet Methods

Method	Method Description
`boolean add(Object o)`	Adds the specified element to this set if it is not already present.
`boolean remove(Object o)`	Removes the specified element from the list if present.
`int size()`	Returns the number of elements in the set.
`void clear()`	Removes all of the elements from this set.
`boolean contains(Object o)`	Returns true if this set contains the specified element.
`boolean isEmpty()`	Returns true if this set contains no elements.

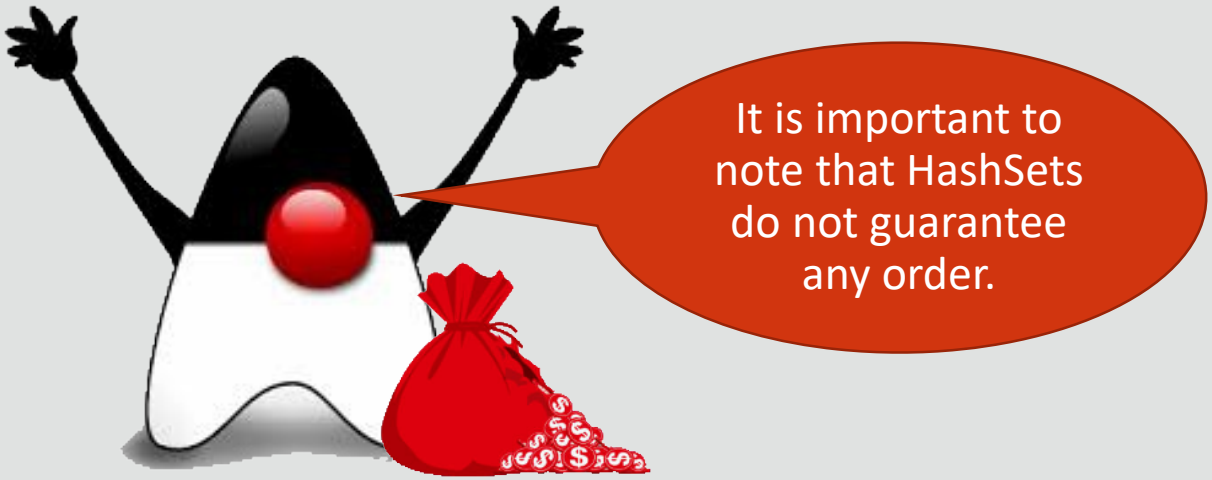
More HashSet methods can be found in the Java API

HashSets Coin Example

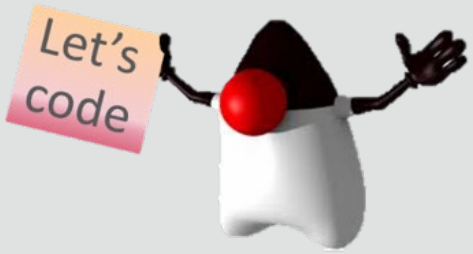
- Imagine there are 35 coins in a bag
- There is no special order to these coins, but the bag can be searched to see if it contains a certain coin
- Coins can be either added or removed from the bag, and the amount of coins is always known
- Think of the bag as the HashSet
- There is no ordering and therefore no indexes of the coins, so we cannot increment through or sort HashSets

HashSets Coin Example

- Even if we incremented through or sorted the coins, with one little shake of the bag, the order is lost
- HashSets have no guarantee that the order will be the same throughout time
- HashSets are often used when you require to tell if an instance that you are trying to add to the collection already exists



HashSets Coin Example



1. Create a project named hashsetexample
2. Create the following Coin class:

```
public class Coin {
   private int denomination;

   public Coin(int denomination){
      this.denomination = denomination;
   }//end constructor

   public int getDenomination(){
      return denomination;
   }//end getDenomination

}//end class Coin
```

HashSets Coin Example

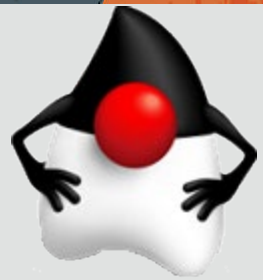
3. Create a test class that has a main method called CoinTestDriver
4. The code below demonstrates the initialization of HashSet bagOfCoins
5. The HashSet bagOfCoins is a set of Coin objects

```
import java.util.HashSet;

public class CoinTestDriver {
   public static void main(String[] args) {
      //create the hashset
      HashSet<Coin> bagOfCoins = new HashSet<Coin>();
   }//end method main

}//end class CoinTestDriver
```

HashSets Coin Task



a) Create a number of coin objects that hold the following denominations (1, 2, 5, 10, 20, 50, 100)

b) The coin objects should be named using the following naming convention:

c) Coindenomination_value (coin5, coin20 etc)

d) Add the coins (1, 2, 5, 10, 20, 50) to the bagOfCoins HashSet. **Do not add coin100!**

e) Add **coin10** to the HashSet twice!

HashSets Coin Task Suggested Solution

```
    HashSet<Coin> bagOfCoins = new HashSet<Coin>();

    //create coin objects
    Coin coin1 = new Coin(1);
    Coin coin2 = new Coin(2);
    Coin coin5 = new Coin(5);
    Coin coin10 = new Coin(10);
    Coin coin20 = new Coin(20);
    Coin coin50 = new Coin(50);
    Coin coin100 = new Coin(100);

    //add the coins to the bag
    bagOfCoins.add(coin1);
    bagOfCoins.add(coin2);
    bagOfCoins.add(coin5);
    bagOfCoins.add(coin10);
    bagOfCoins.add(coin20);
    bagOfCoins.add(coin50);
    bagOfCoins.add(coin10);
}//end method main
```

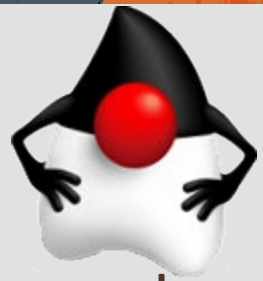
Searching Through HashSets

- Although HashSets do not have ordering, we can search through them, just like we could search through the bag to see if the coin we are looking for is in the bag
- For example, to search for a coin in the HashSet bagOfCoins, use HashSet's contains(element) as demonstrated below:

```
bagOfCoins.contains(coin5));
//returns true if bagOfCoins contains the coin
```

HashSets are quicker at finding elements than Lists especially as the list grows in size. The HashSet does not contain duplicates so it will never be larger than the list.

HashSets Coin Task continued



f) Create a static method named findCoin that accepts the HashSet and a coin object as parameters and does not return any values

g) The method should be called from main

h) Use an if statement to check if the bag contains the coin that was passed as the parameter. If the coin is in the bag then the following output should be displayed:

```
There is a coin10 in the bag
```

otherwise

```
There is no coin10 in the bag
```

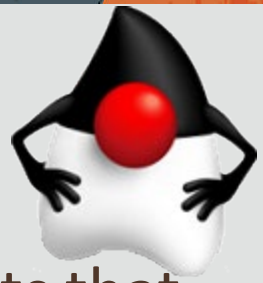
HashSets Coin Task Suggested Solution

- Test it by using coin10 and coin100 for the search

```
      findCoin(bagOfCoins, coin10);
      findCoin(bagOfCoins, coin100);
   }//end method main

   static void findCoin(HashSet<Coin> bagOfCoins, Coin coin){
      if(bagOfCoins.contains(coin))
         System.out.println("There is a coin " +
                            coin.getDenomination() + " in the bag");
      else
         System.out.println("There is no coin " +
                            coin.getDenomination() + " in the bag");
   }//end method findCoin
}//end class CoinTestDriver
```

HashSets Coin Task continued



i) Create a static method named displayBagContents that accepts the HashSet as a parameter and does not return any values

j) The method should be called from main

k) Use an enhanced for loop to display the denomination value for the coins stored in the HashSet

l) Create a static method named displayBagDetails that accepts the HashSet as a parameter and does not return any values

m) If the HashSet is empty it should display a message stating the bag is empty, otherwise display a how many coins are in the bag

HashSets Coin Task Suggested Solution

```
        findCoin(bagOfCoins, coin100);
        displayBagContents(bagOfCoins);
        displayBagDetails(bagOfCoins);
    }//end method main

    static void displayBagContents(HashSet<Coin> bagOfCoins){
        //display the contents of the bag
        for(Coin coin: bagOfCoins)
            System.out.println(coin.getDenomination());
        //end for
    }//end method displayBagContents

    static void displayBagDetails(HashSet<Coin> bagOfCoins){
        if(bagOfCoins.isEmpty())
            System.out.println("There are no coins in the bag");
        else
            System.out.println("There are " + bagOfCoins.size()
                               + " coins in the bag");
    }//end method displayBagDetails
```

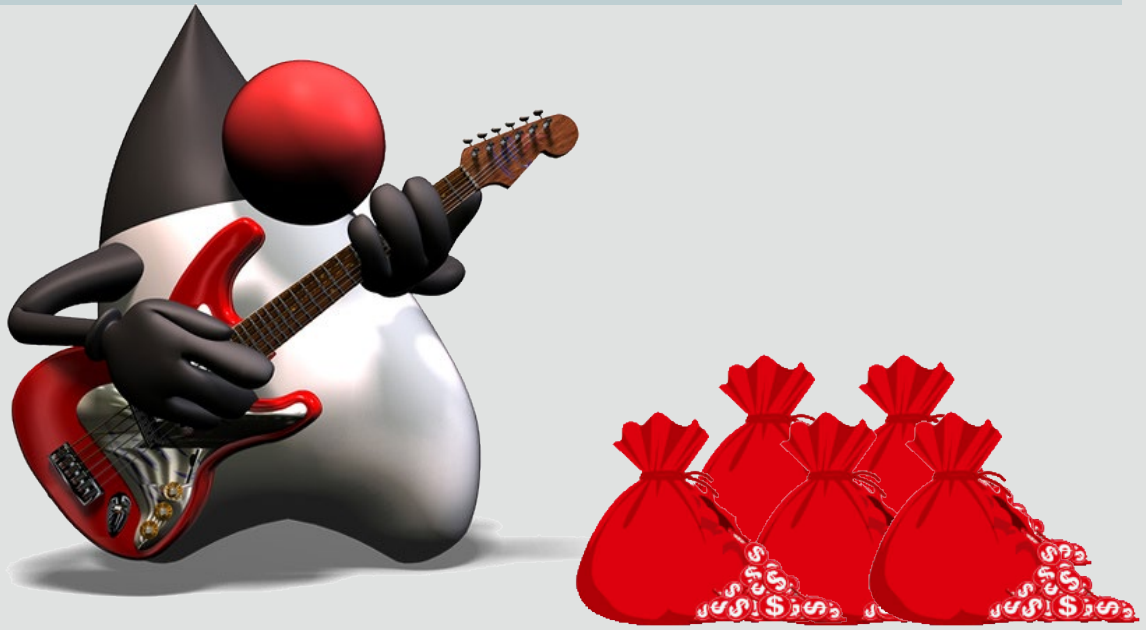
HashSets Coin Task continued

n) Identify the HashSet method that removes all of the elements from the set and write it directly under the displayBagDetails() method call in main

o) Call displayBagDetails() again, after you have removed all of the elements from the set in main

HashSets Coin Task Suggested Solution

```
    findCoin(bagOfCoins, coin10);
    findCoin(bagOfCoins, coin100);
    displayBagContents(bagOfCoins);
    displayBagDetails(bagOfCoins);
    bagOfCoins.clear();
    displayBagDetails(bagOfCoins);
}//end method main
```



Terminology

- Key terms used in this lesson included:
  - Generics
  - Collection
  - List
  - ArrayList
  - Set
  - HashSet

Summary

- In this lesson, you should have learned how to:
  - Create a collection without using generics
  - Create a collection using generics
  - Implement an ArrayList
  - Implement a HashSet


ORACLE
Academy