Index

[Exercise 1: Generic Programming 1 3](#_Toc65835624)

[Task 1 - Create Box Class 3](#_Toc65835625)

[Task 2 - Create a running Program 3](#_Toc65835626)

[Task 3 - Execute your program 3](#_Toc65835627)

[Exercise 2: Generic Programming 2 4](#_Toc65835628)

[Task 1 - Using <T extends A> syntax to restrict T to be a subclass of A 4](#_Toc65835629)

[Task 2 - Try with Unbound wildcard 6](#_Toc65835630)

[Task 3 - Try with lower bound wildcard 6](#_Toc65835631)

[Task 4 - Try wild higher bound wildcard 7](#_Toc65835632)

[Exercise 3: ArrayList 7](#_Toc65835633)

[Task 1 - Create ExArrayList class 7](#_Toc65835634)

[Task 2 - Write code for main() method 7](#_Toc65835635)

[Task 3 - Execute your program 8](#_Toc65835636)

[Exercise 4: Student List 9](#_Toc65835637)

[Task 1 - Write code for Student class 9](#_Toc65835638)

[Task 2 - Write code for People class 9](#_Toc65835639)

[Task 3 - Create a running Program 10](#_Toc65835640)

[Task 4 - Execute your program 11](#_Toc65835641)

[Exercise 5: LinkedList 11](#_Toc65835642)

[Task 1 - Create ExLinkedList class extending Thread class 11](#_Toc65835643)

[Task 2 - Write code for main() method 11](#_Toc65835644)

[Task 3 - Execute your program 14](#_Toc65835645)

[Exercise 6: Set 15](#_Toc65835646)

[Task 1 - Create ListDirectoryRecusive class extending Thread class 15](#_Toc65835647)

[Task 2 - Write code for main() method 15](#_Toc65835648)

[Task 3 - Execute your program 16](#_Toc65835649)

[Exercise 7: Map 17](#_Toc65835650)

[Task 1 - Create exMap class extending Thread class 17](#_Toc65835651)

[Task 2 - Write code for main() method 17](#_Toc65835652)

[Task 3 - Execute your program 19](#_Toc65835653)

[Exercise 8: Employee Management 19](#_Toc65835654)

[Task 1 - Create Employee Class 19](#_Toc65835655)

[Task 2 - Create HREmpoyee Class 20](#_Toc65835656)

[Task 3 - Create a running Program 20](#_Toc65835657)

[Task 4 - Execute your program 21](#_Toc65835658)

[Task 5 - Create EmployeeDetail Class 21](#_Toc65835659)

[Task 6 - Edit Running Program 21](#_Toc65835660)

[Task 7 - Execute your program 22](#_Toc65835661)

1. Generic Programming 1

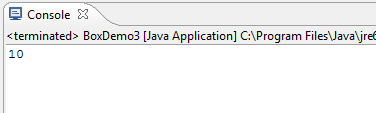
## Create Box Class

* 1. **public class** Box<T> {
  2. **private** T t; // T stands for "Type"
  3. Box() {
  4. t=null;
  5. }
  6. Box(T t) {
  7. this.t=t;
  8. }
  9. **public** void set(T t) {
  10. this.t = t;
  11. }
  12. **public** T get() {
  13. return t;
  14. }
  15. }

## Create a running Program

* 1. **public** **class** BoxDemo {
  2. **public** **static** **void** main(String[] args) {
  3. Box<Integer> integerBox = **new** Box<Integer>();
  4. integerBox.set(**new** Integer(10));
  5. Integer someInteger = integerBox.get();
  6. System.*out*.println(someInteger);
  7. }
  8. }

## Execute your program



1. Generic Programming 2

## Using <T extends A> syntax to restrict T to be a subclass of A

### Write class A

* 1. public class A {
  2. public String toString() {
  3. return "Class A";
  4. }
  5. }

### Write class B

* 1. public class B extends A{
  2. public String toString() {
  3. return "Class B";
  4. }
  5. }

### Write class C

* 1. public class C extends B{
  2. public String toString() {
  3. return "Class C";
  4. }
  5. }

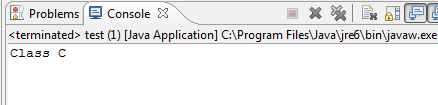
### Edit class Box to work with class B and C

* 1. public class Box<T extends B> {
  2. private T t;
  3. Box() {
  4. t=null;
  5. }
  6. Box(T t) {
  7. this.t=t;
  8. }
  9. public void set(T t) {
  10. this.t = t;
  11. }
  12. public T get() {
  13. return t;
  14. }
  15. }

### Test with class C

* 1. public class test {
  2. public static void main(String[] args) {
  3. Box<C> myBox = new Box<C>();
  4. myBox.set(new C());
  5. B b = myBox.get();
  6. System.*out*.println(b);
  7. }
  8. }

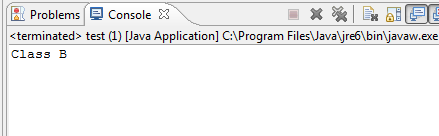
=>Run:



### Test with class B

* 1. public class test {
  2. public static void main(String[] args) {
  3. Box<B> myBox = new Box<B>();
  4. myBox.set(new B());
  5. B b = myBox.get();
  6. System.*out*.println(b);
  7. }
  8. }

=>Run:



### Test with class A

* 1. public class test {
  2. public static void main(String[] args) {
  3. Box<A> myBox = new Box<A>();
  4. myBox.set(new A());
  5. A a = myBox.get();
  6. System.*out*.println(a);
  7. }
  8. }

## Try with Unbound wildcard

### Using Wildcard

* 1. Box<?> myBox;
  2. myBox = **new** Box<B>(new B());

### Write code to test method **get** and **set**

* 1. //Your code

### Edit Box class:

* 1. public class Box<T **extends** B> {
  2. ..
  3. }

### Write code to test method **get** and **set (get Object and can’t set)**

## Try with lower bound wildcard

### Using Wildcard

* 1. Box<? extends B> myBox;
  2. myBox = **new** Box<B>(new B());

### Write code to test method **get** and **set**

* 1. //Your code

## Try wild higher bound wildcard

### Using Wildcard

* 1. Box<? super B> myBox;
  2. myBox = **new** Box<B>(new B());

### Write code to test method **get** and **set**

1. ArrayList

## Create ExArrayList class

### Import Libraries

* 1. **import** java.util.ArrayList;

## Write code for main() method

### Creates a new List called arrayList.

* 1. ArrayList<String> list = **new** ArrayList<String>();

### Adds Five fruit elements into arrayList and prints it.

* 1. list.add("apples");
  2. list.add("bananas");
  3. list.add("grapes");
  4. list.add("pears");
  5. list.add("black plums");
  6. System.*out*.println("My array list currently includes: " + list);

### Adds the elements onto the list and print them

* 1. list.add(0, "watermelons");
  2. System.*out*.println("My array list now includes: " + list);
  3. list.add(3, "oranges");
  4. System.*out*.println("My array list now includes: " + list);
  5. System.*out*.println("'grapes' is an element in the list: "
  6. + list.contains("grapes"));
  7. System.*out*.println("'apricots' is an element in the list: "
  8. + list.contains("apricots"));
  9. System.*out*.println("Element at index 2 is: " + list.get(2));
  10. System.*out*.println("Element at index 1 is: " + list.get(1));
  11. System.*out*.println("The array list is currently empty: " + list.isEmpty());

### Return the current size of the array.

* 1. System.*out*.println("The size of our current array list is: " + list.size());

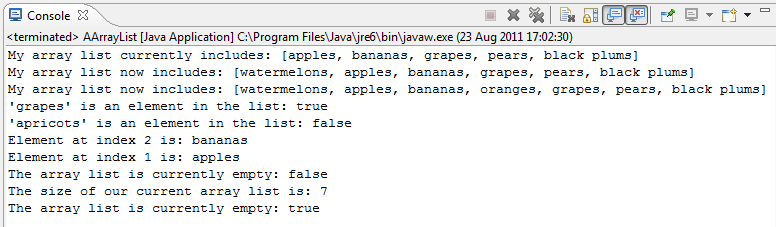
### Clear the entire array. (Removes all elements within it)

* 1. list.clear();

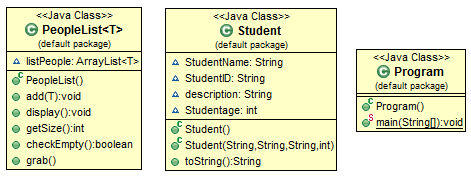
### Check to see if list is empty or not.

* 1. System.*out*.println("The array list is currently empty: " + list.isEmpty());

## Execute your program

* 1. 

1. Student List



## Write code for Student class

* 1. **public** **class** Student {
  2. String StudentName;
  3. String StudentID;
  4. String description;
  5. **int** Studentage;
  6. **public** Student() {
  7. }
  8. **public** Student(String pName, String pID, String descr, **int** age) {
  9. StudentName = pName;
  10. StudentID = pID;
  11. description = descr;
  12. Studentage = age;
  13. }
  14. **public** String toString() {
  15. **return** "Student Name: " + StudentName + "\tStudent ID: " + StudentID
  16. + "\tDescription: " + description + "\tage: " + Studentage;
  17. }
  18. }

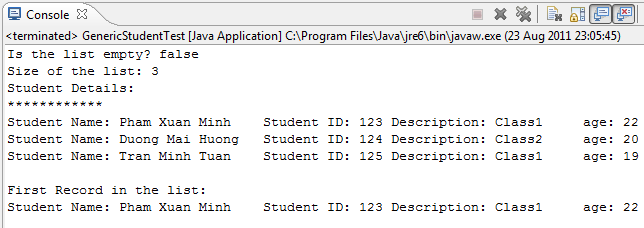
## Write code for People class

* 1. **public** **class** PeopleList<T> {
  2. ArrayList<T> listPeople;
  3. **public** PeopleList() {
  4. listPeople = **new** ArrayList<T>();
  5. }
  6. **public** **void** add(T obj) {
  7. listPeople.add(obj);
  8. }
  9. **public** **void** display() {
  10. **for** (T objPeople : listPeople) {
  11. System.*out*.println(objPeople.toString());
  12. }
  13. }
  14. **public** **int** getSize() {
  15. **return** listPeople.size();
  16. }
  17. **public** **boolean** checkEmpty() {
  18. **return** (listPeople.size() == 0);
  19. }
  20. **public** T grab() {
  21. **if** (!checkEmpty()) {
  22. **return** listPeople.remove(0);
  23. } **else**
  24. **return** **null**;
  25. }
  26. }

## Create a running Program

* 1. **public** **class** Program {
  2. **public** **static** **void** main(String[] args) **throws** Exception {
  3. PeopleList<Student> objStudent = **new** PeopleList<Student>();
  4. objStudent.add(**new** Student("Pham Xuan Minh", "123", "Class1", 22));
  5. objStudent.add(**new** Student("Duong Mai Huong", "124", "Class2", 20));
  6. objStudent.add(**new** Student("Tran Minh Tuan", "125", "Class1", 19));
  7. System.*out*.println("Is the list empty? " + objStudent.checkEmpty());
  8. System.*out*.println("Size of the list: " + objStudent.getSize());
  9. System.*out*.println("Student Details:");
  10. System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*");
  11. objStudent.display();
  12. }
  13. }

## Execute your program



1. LinkedList

## Create ExLinkedList class extending Thread class

### In the "Package Explorer" (left panel) ⇒ Right-click on your Project (or use the "File" menu) ⇒ New ⇒ Class.

### The "New Java Class" dialog pops up.

* In "Name" field, enter “ExLinkedList".
* Check "public static void main(String[] args)" box.
* Click "Finish".

### Import Libraries

* 1. **import** java.util.Iterator;
  2. **import** java.util.LinkedList;

## Write code for main() method

### Create a LinkedList and other variables

* 1. LinkedList <Integer>list = **new** LinkedList<Integer>();
  2. **int** num1 = 11, num2 = 22, num3 = 33, num4 = 44;
  3. **int** size;
  4. Iterator iterator;

### Adding data in the list

* 1. list.add(num1);
  2. list.add(num2);
  3. list.add(num3);
  4. list.add(num4);
  5. size = list.size();
  6. System.*out*.print( "Linked list data: ");

### Create a iterator

* 1. iterator = list.iterator();
  2. **while** (iterator.hasNext()){
  3. System.*out*.print(iterator.next()+" ");
  4. }
  5. System.*out*.println();

### Check list empty or not

* 1. **if** (list.isEmpty()){
  2. System.*out*.println("Linked list is empty");
  3. }
  4. **else**{
  5. System.*out*.println( "Linked list size: " + size);
  6. }
  7. System.*out*.println("Adding data at 1st location: 55");

### Add and print data in list

* 1. list.addFirst(55);
  2. System.*out*.print("Now the list contain: ");
  3. iterator = list.iterator();
  4. **while** (iterator.hasNext()){
  5. System.*out*.print(iterator.next()+" ");
  6. }
  7. System.*out*.println();
  8. System.*out*.println("Now the size of list: " + list.size());
  9. System.*out*.println("Adding data at last location: 66");
  10. list.addLast(66);
  11. System.*out*.print("Now the list contain: ");
  12. iterator = list.iterator();
  13. **while** (iterator.hasNext()){
  14. System.*out*.print(iterator.next()+" ");
  15. }
  16. System.*out*.println();
  17. System.*out*.println("Now the size of list: " + list.size());
  18. System.*out*.println("Adding data at 3rd location: 55");
  19. list.add(2,99);
  20. System.*out*.print("Now the list contain: ");
  21. iterator = list.iterator();
  22. **while** (iterator.hasNext()){
  23. System.*out*.print(iterator.next()+" ");
  24. }
  25. System.*out*.println();
  26. System.*out*.println("Now the size of list: " + list.size());

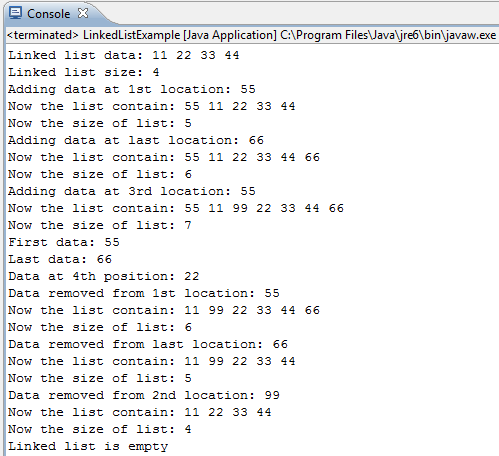
### Retrieve data

* 1. System.*out*.println("First data: " + list.getFirst());
  2. System.*out*.println("Last data: " + list.getLast());
  3. //Retrieve specific data
  4. System.*out*.println("Data at 4th position: " + list.get(3));

### Remove data

* 1. **int** first = list.removeFirst();
  2. System.*out*.println("Data removed from 1st location: " + first);
  3. System.*out*.print("Now the list contain: ");
  4. iterator = list.iterator();
  5. //After removing data
  6. **while** (iterator.hasNext()){
  7. System.*out*.print(iterator.next()+" ");
  8. }
  9. System.*out*.println();
  10. System.*out*.println("Now the size of list: " + list.size());
  11. //Remove last
  12. **int** last = list.removeLast();
  13. System.*out*.println("Data removed from last location: " + last);
  14. System.*out*.print("Now the list contain: ");
  15. iterator = list.iterator();
  16. //After removing data
  17. **while** (iterator.hasNext()){
  18. System.*out*.print(iterator.next()+" ");
  19. }
  20. System.*out*.println();
  21. System.*out*.println("Now the size of list: " + list.size());
  22. //Remove 2nd data
  23. **int** second = list.remove(1);
  24. System.*out*.println("Data removed from 2nd location: " + second);
  25. System.*out*.print("Now the list contain: ");
  26. iterator = list.iterator();
  27. //After removing data
  28. **while** (iterator.hasNext()){
  29. System.*out*.print(iterator.next()+" ");
  30. }
  31. System.*out*.println();
  32. System.*out*.println("Now the size of list: " + list.size());
  33. //Remove all
  34. list.clear();
  35. **if** (list.isEmpty()){
  36. System.*out*.println("Linked list is empty");
  37. }
  38. **else**{
  39. System.*out*.println( "Linked list size: " + size);
  40. }

## Execute your program

* 1. 

1. Set

## Create ListDirectoryRecusive class extending Thread class

### Import Libraries

* 1. **import** java.util.HashSet;
  2. **import** java.util.Set;
  3. **import** java.util.TreeSet;

## Write code for main() method

### Creates a HashSet called s

* 1. Set<String> s = **new** HashSet<String>();

### Adds elements into s and prints it.

* 1. s.add("grapes");
  2. s.add("bananas");
  3. s.add("apples");
  4. s.add("pears");
  5. s.add("black plums");
  6. System.*out*.println("My set currently includes: " + s);
  7. s.add("watermelons");
  8. System.*out*.println("My set now includes: " + s);
  9. s.add("bananas");
  10. System.*out*.println("My set now includes: " + s);
  11. // Determines whether or not 'grapes' is in the s.
  12. System.*out*.println("'grapes' is an element in the set:
  13. s.contains("grapes"));
  14. // Determines whether or not 'apricots' is in the s.
  15. System.*out*.println("'apricots' is an element in the set:
  16. s.contains("apricots"));
  17. // Checks to see if s is empty or not.
  18. System.*out*.println("The set is currently empty: " + s.isEmpty());
  19. // Returns the current size of the array.
  20. System.*out*.println("The size of set is: " + s.size());

### Copy HashSet s to a TreeSet

* 1. Set<String> sortedSet = **new** TreeSet<String>(s);
  2. System.*out*.println("My sorted set now includes: " + sortedSet);

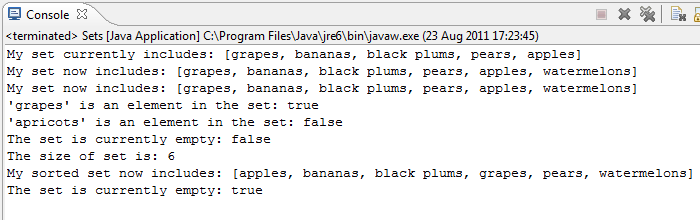
### Clears the entire array. (Removes all elements within it)

* 1. s.clear();

### Checks to see if s is empty or not.

* 1. System.*out*.println("The set is currently empty: " + s.isEmpty());

## Execute your program

* 1. 

1. Map

## Create exMap class extending Thread class

### Import Libraries

* 1. **import** java.util.HashMap;
  2. **import** java.util.TreeMap;

## Write code for main() method

### Creates a HashMap called myMap This HashMap contains many pairs of keys and values Strings are keys, Integers are values The keys must be unique (as a set must be)

* 1. HashMap<String, Integer> myMap = **new** HashMap<String, Integer>();

### Putting pairs of keys and values

* 1. // (int) (Math.random() \* 10) gives a random integer from 0-9
  2. myMap.put("Apples", (**int**) (Math.*random*() \* 10));
  3. // Note that although myMap requires an Integer, int will work too
  4. // this is a feature of Java 5.0 called auto-boxing
  5. // If an int is required, an Integer will also work: auto-unboxing
  6. myMap.put("Bananas", (**int**) (Math.*random*() \* 10));
  7. myMap.put("Grapes", (**int**) (Math.*random*() \* 10));
  8. // Cherries has the same Integer value as Grapes
  9. myMap.put("Cherries", myMap.get("Grapes"));
  10. myMap.put("Watermelons", (**int**) (Math.*random*() \* 10));
  11. // Since the keys must be unique, there will only be one
  12. // "Watermelons" key
  13. myMap.put("Watermelons", (**int**) (Math.*random*() \* 10));
  14. // Prints out keys and values
  15. System.*out*.println("my map is: " + myMap);

### Creates a SortedMap of myMap, sorted by the String keys

* 1. TreeMap<String, Integer> mySortedMap = **new** TreeMap<String, Integer>(myMap);
  2. System.*out*.println("my sorted map is: " + mySortedMap);
  3. HashMap<Integer, String> myMap2 = **new** HashMap<Integer, String>();

### Exchanges keys and values from myMap to myMap2

* 1. // keySet() contains all keys in myMap
  2. **for** (String s : myMap.keySet())
  3. myMap2.put(myMap.get(s), **new** String(s));
  4. // Note that myMap2 is smaller than myMap
  5. System.*out*.println("my second map is: " + myMap2);

### Creates a SortedMap of myMap2, sorted by the Integer keys

* 1. TreeMap<Integer, String> mySortedMap2 = **new** TreeMap<Integer,String>(myMap2);
  2. System.*out*.println("my second sorted map is: " + mySortedMap2);
  3. System.*out*.println();

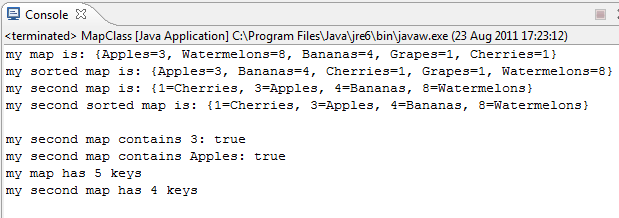
### Checks if 3 is a key

* 1. System.*out*.println("my second map contains 3: " + myMap2.containsKey(3));

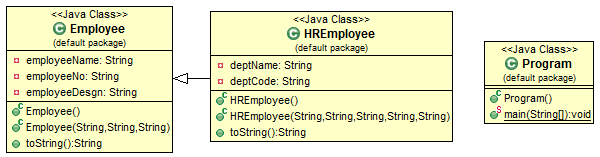
### Checks if "Apples" is a value

* 1. System.*out*.println("my second map contains Apples: "
  2. + myMap2.containsValue("Apples"));
  3. System.*out*.println("my map has " + myMap.size() + " keys");
  4. System.*out*.println("my second map has " + myMap2.size() + " keys");

## Execute your program

* 1. 

1. Employee Management



## Create Employee Class

* 1. **public** **class** Employee {
  2. **private** String employeeName;
  3. **private** String employeeNo;
  4. **private** String employeeDesgn;
  5. **public** Employee() {
  6. }
  7. **public** Employee(String eName, String eNo, String eDesgn) {
  8. employeeName = eName;
  9. employeeNo = eNo;
  10. employeeDesgn = eDesgn;
  11. }
  12. **public** String toString() {
  13. **return** "Name: " + employeeName + "\tNumber: " + employeeNo
  14. + "\tDesignation: " + employeeDesgn;
  15. }
  16. }

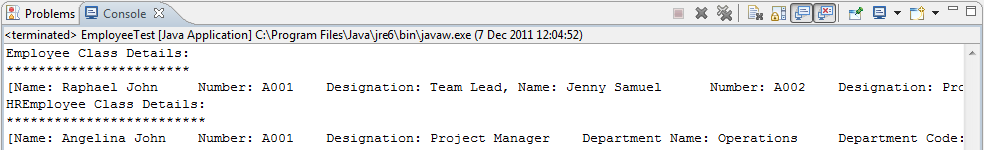
## Create HREmpoyee Class

* 1. **public** **class** HREmployee **extends** Employee {
  2. **private** String deptName;
  3. **private** String deptCode;
  4. **public** HREmployee() {
  5. }
  6. **public** HREmployee(String name, String num, String desig, String dept,
  7. String code) {
  8. **super**(name, num, desig);
  9. deptName = dept;
  10. deptCode = code;
  11. }
  12. **public** String toString() {
  13. **return** **super**.toString() + "\tDepartment Name: " + deptName
  14. + "\tDepartment Code:" + deptCode;
  15. }
  16. }

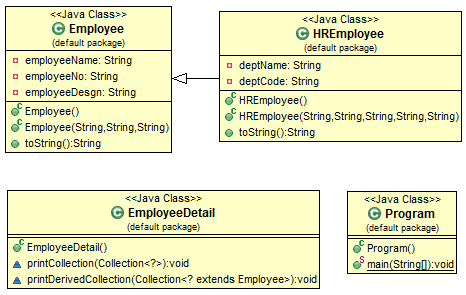
## Create a running Program

* 1. **public** **class** EmployeeTest {
  2. **public** **static** **void** main(String[] args) {
  3. ArrayList<Employee> emp = **new** ArrayList<Employee>();
  4. System.*out*.println("Employee Class Details: ");
  5. System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");
  6. emp.add(0, **new** Employee("Raphael John", "A001", "Team Lead"));
  7. emp.add(1, **new** Employee("Jenny Samuel", "A002", "Project Manager"));
  8. System.*out*.print(emp);
  9. /\*\*
  10. \* Creates an user defined ArrayList of type HREmployee.
  11. \*/
  12. ArrayList<HREmployee> empHR = **new** ArrayList<HREmployee>(1);
  13. System.*out*.println("\nHREmployee Class Details: ");
  14. System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");
  15. empHR.add(0, **new** HREmployee("Angelina John", "A001", "Project Manager",
  16. "Operations", "01"));
  17. empHR.add(1, **new** HREmployee("Susan Pitt", "A002", "Team Leader",
  18. "Writing", "02"));
  19. System.*out*.print(empHR);
  20. }
  21. }

## Execute your program



In above next task, we will improve your program by adding the new EmployeeDetail Class



## Create EmployeeDetail Class

* 1. **public** **class** EmployeeDetail {
  2. **public** EmployeeDetail() {
  3. }
  4. **void** printCollection(Collection<?> collect) {
  5. **for** (Object obj : collect)
  6. System.*out*.println(obj.toString());
  7. }
  8. **void** printDerivedCollection(Collection<? **extends** Employee> collect) {
  9. **for** (Object obj : collect)
  10. System.*out*.println(obj.toString());
  11. }
  12. }

## Edit Running Program

* 1. **public** **class** EmployeeTest {
  2. **public** **static** **void** main(String[] args) {
  3. ArrayList<Employee> emp = **new** ArrayList<Employee>(1);
  4. System.*out*.println("Employee Class Details: ");
  5. System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");
  6. emp.add(0, **new** Employee("Raphael John", "A001", "Team Lead"));
  7. emp.add(1, **new** Employee("Jenny Samuel", "A002", "Project Manager"));
  8. **EmployeeDetail empObj = new EmployeeDetail();**
  9. **empObj.printCollection(emp);**
  10. /\*\*
  11. \* Creates an user defined ArrayList of type HREmployee.
  12. \*/
  13. ArrayList<HREmployee> empHR = **new** ArrayList<HREmployee>(1);
  14. System.*out*.println("\nHREmployee Class Details: ");
  15. System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");
  16. empHR.add(0, **new** HREmployee("Angelina John", "A001", "Project Manager",
  17. "Operations", "01"));
  18. empHR.add(1, **new** HREmployee("Susan Pitt", "A002", "Team Leader",
  19. "Writing", "02"));
  20. **empObj.printDerivedCollection(empHR);**
  21. }
  22. }

## Execute your program

