1 god 4 all) <https://java2blog.com/core-java-tutorial-for-beginners-experienced/bvnjava8> 2 ocen all god

1 god 4 all) <http://javasampleapproach.com/java/java-8-date-time> bvn bvn java8 to ocean all god

1 god) <https://www.logicbig.com/tutorials/core-java-tutorial/java-multi-threading/executor-framework.html> bvn bvn java8 to ocean all god

1 god) <http://javasampleapproach.com/hibernate/use-hibernate-lazy-fetch-eager-fetch-type-spring-boot-mysql> bvn bvn java8 to ocean all god

1 god) <http://www.vogella.com/tutorials/java.html> bvn bvn java8 to ocean all god

<https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/package-summary.html> bvn

1bvn)<http://www.java2s.com/> ###\*\*\*VVV god java ---- lrn java full from here 1st

1vvv)<https://codescracker.com/java/java-interview-questions.htm> lrn the best coding logics from here

1bvn)<http://www.jcombat.com/core-java/introduction-to-executor-framework> bvn java ocean all god

1vvv) <https://www.concretepage.com/> bvn java8 to ocean all god

1vvv )<http://www.novixys.com/blog/java-hashmap-examples/> lrn java still from here 1st

1v)<http://javajee.com/comparable-and-comparator-interfaces> java god bvn theory best

2v)<https://beginnersbook.com/2017/08/comparable-interface-in-java-with-example/> java god bvn easy

<http://www.java67.com/2015/01/how-to-sort-hashmap-in-java-based-on.html> java god bvn

1. https://github.com/eugenp/tutorials/tree/master/core-java
2. <http://www.codejava.net/java-core/concurrency> ###\*\*\*VVV god java , spring xml …
3. <http://www.javamadesoeasy.com/?m=1> ###\*\*\*VVV god java
4. http://www.baeldung.com/java-bubble-sort

<http://javarevisited.blogspot.in/2011/06/comparator-and-comparable-in-java.html>

<https://www.programcreek.com/2013/03/java-sort-map-by-value/>

<https://github.com/ryanlr/Design-Patterns/tree/master/src/com/programcreek/designpatterns>

<http://www.instanceofjava.com/p/collections.html> bvn prac

3)<https://www.programcreek.com/> java AI,DS,Algo’s BVN god

Java 8 coding practicing;

4)http://www.logicbig.com/tutorials/core-java-tutorial/java-multi-threading/scheduled-executor/

<http://www.technicalkeeda.com/tutorials/java-8-tutorials> vv

<https://github.com/ryanlr/Java8Lambda> gods of gods

https://www.programcreek.com/simple-java-8-lambdas/ gods of gods

<https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html>

<http://tamanmohamed.blogspot.in/> bvn JavaSE8-Features

<https://github.com/mohamed-taman/JavaSE8-Features/> vvvvvvv)

<https://blogs.oracle.com/java/java-9-3> orcle site

**What's new in JAVA 8?**  
<http://javatechnologycenter.com/question/Java_8newfeatures/overview.php> bvn

JAVA 8 is a major feature release of JAVA programming language development. It's initial version was released on 18th March'14. With Java 8 release, java provided supports for functional programming, new JavaScript engine, new APIs for date time manipulation, new streaming api etc.  
  
**New features**

* **Lambda expression** - Adds functional processing capability to JAVA.
* **Method references** - Referencing functions by their name instead of invoking them directly. Using functions as parameter.
* **Default method** - Interface to have default method implementation.
* **New Tools** - New compiler tools and utilities are added like jdeps to figure out the dependencies.
* **Stream API** - New stream API to facilitate pipeline processing.
* **Date Time API** - Improved date time api.
* **Optional** - Emphasis on best practices, to handle null values properly.
* **Nashorn , JavaScript Engine** - A JAVA based engine to execute JavaScript code.

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class Example {

public static void main(String[] args) {

List<Integer> test = Arrays.asList(1,2,3,4,5,6,7,8,9,10);

System.out.println(businessLogic(test));

System.out.println(businessLogicFunctional(test));

}

public static List<Integer> businessLogic(List<Integer> nums) {

List<Integer> res = new ArrayList();

for(int i=0;i<nums.size();i++) {

int num = nums.get(i);

if (num % 2 == 0) {

int temp = num + 100;

if (temp % 10 == 0) {

res.add(temp);

}

}

}

return res;

}

public static List<Integer> businessLogicFunctional(List<Integer> nums) {

return nums.stream()

.filter(i -> i % 2 == 0)

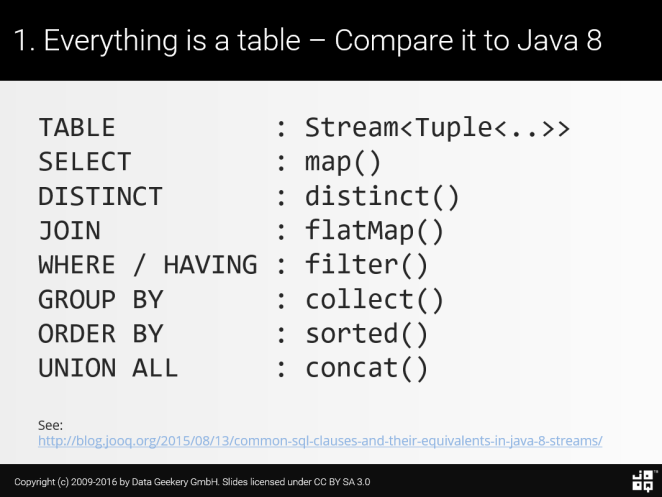
.map(i -> i + 100)

.filter(i -> i % 10 == 0)

.collect(Collectors.toList());

}

}



equals() method is used to [compare Objects](http://javarevisited.blogspot.sg/2011/06/comparator-and-comparable-in-java.html) for equality while hashCode is used to generate an integer code corresponding to that object.  
  
equals and hashCode have used extensively in Java core library like they are used while inserting and retrieving Object in HashMap. equals method is also used to avoid duplicates on HashSet and other Set implementation and every other place where you need to compare Objects.   
note: Default implementation of equals() class provided by java.lang.Object compares memory location and only return true if two reference variable is pointing to same memory location   
Java recommends to override equals and hashCode method if equality is going to be defined by logical way or via some business logic   
Since [HashMap and Hashtable](http://javarevisited.blogspot.sg/2010/10/difference-between-hashmap-and.html) in Java rely on equals() and hashCode() method for comparing keys and values, Java provides following rules to override equals method Java. As per following rule equals method in Java should be:

1) **Reflexive :** Object must be equal to itself.

2) **Symmetric :** if a.equals(b) is true then b.equals(a) must be true.

3) **Transitive :** if a.equals(b) is true and b.equals(c) is true then c.equals(a) must be true.

4) **Consistent :** multiple invocations of equals() method must return the same value until any of properties are modified. So if two objects are equals in Java they will remain equals until any of their property is modified.

5) **Null comparison :** comparing any object to null must be false and should not result in NullPointerException. For example a.equals(null) must be false, passing unknown object, which could be null,  to equals in Java is is actually a Java coding [best practice to avoid NullPointerException in Java](http://javarevisited.blogspot.sg/2012/06/common-cause-of-javalangnullpointerexce.html).

## Equals and hashCode contract in Java

And equals method in Java must follow its contract with hashcode method in Java as stated below.

1) If two objects are equal by equals() method then there hashcode must be same.

2) If two objects are not equal by equals() method then there hashcode could be same or different.

A sorted set is a set with ordering on its elements.

SortedSet interface represents a sorted set in Java Collection Framework.

The elements in a SortedSet can be sorted in a natural order with Comparable interface or using a Comparator.

A SortedSet must know how to sort its elements as they are added by checking two interfaces:

* If its elements implement the Comparable interface, it will use the compareTo() method to sort items. We can call this as sorting in natural order.
* We can passin a Comparator to do custom sorting.

If a Comparator is specified, the Comparator is used for sorting and ignore the Comparableinterface.

The TreeSet class is an implementation for the SortedSet interface in the Collections Framework.

**Comparators and comparable** in Java are two interfaces which is used to implement sorting in Java. Comparator should be used as an utility to sort objects which Comparable should be provided by default.  
the logical difference between these two is *Comparator in Java* compare two objects provided to him, while Comparable interface compares "this" reference with the object specified.  
4) Comparable in Java is used to implement **natural ordering of object**. In Java API String, Date and wrapper classes implements Comparable interface.Its always good practice to override compareTo() for value objects.

5) If any class implement Comparable interface in Java then collection of that object either [List](http://javarevisited.blogspot.sg/2012/04/difference-between-list-and-set-in-java.html) or Array can be sorted automatically by using  Collections.sort() or Arrays.sort() method and object will be sorted based on there natural order defined by CompareTo method.

6)Objects which implement *Comparable in Java*  can be used as keys in a SortedMap like [TreeMap](http://javarevisited.blogspot.sg/2011/12/treemap-java-tutorial-example-program.html) or elements in a SortedSet  for example TreeSet, without specifying any Comparator.

So in Summary if you want to **sort objects based on natural order** then use Comparable in Java and if you want to sort on some other attribute of object then use Comparator in Java. Now to understand these concepts lets see an example or real life coding:  
  
  
**Comparable**

A comparable object is capable of comparing itself with another object. The class itself must implements the java.lang.Comparable interface in order to be able to compare its instances.

**Comparator**

A comparator object is capable of comparing two different objects. The class is not comparing its instances, but some other class’s instances. This comparator class must implement the java.util.Comparator interface.

A **comparable** object is capable of comparing itself with another object. The class itself must implements the **java.lang.Comparable** interface to compare its instances. and we override the method compareTo() of Comparable interface.

Unlike Comparable, **Comparator** is external to the element type we are comparing. It’s a separate class. We create multiple separate classes (that implement Comparator) to compare by different members.

Collections class has a second sort() method and it takes Comparator. The sort() method invokes the compare() to sort objects.

Comparable and comparator:

**Comparable interface** is mainly used to sort the arrays (or lists) of **custom objects**. but we can sort elements of arrays and Wrapper classes that already implements Comparable.

Ex: Arrays sorting:

int[] arr = {11,55,22,0,89};

Arrays.sort(arr);

Sorting string arrays:

String[] names = {"Steve", "Ajeet", "Kyle"};

Arrays.sort(names);

Collections sorting(since fruits names nothing but strings):

Collections.sort(fruits);

**However if you want to sort the objects of custom class then you need to implement the Comparable interface in our custom class.**

This interface has only one method which is:

public abstract int compareTo(T obj)

ex: public class Author implements Comparable<Author>

String firstName;

String lastName;

String bookName;

@Override

public int compareTo(Author au){

return this.lastName.compareTo(au.lastName);

}

Sorting:

ArrayList<Author> al=new ArrayList<Author>();

al.add(new Author("Henry","Miller", "Tropic of Cancer"));

al.add(new Author("Nalo","Hopkinson", "Brown Girl in the Ring"));

al.add(new Author("Frank","Miller", "300"));

al.add(new Author("Deborah","Hopkinson", "Sky Boys"));

al.add(new Author("George R. R.","Martin", "Song of Ice and Fire"));

/\*

\* Sorting the list using Collections.sort() method, we

\* can use this method because we have implemented the

\* Comparable interface in our user defined class Author

\*/

Collections.sort(al);

Note:We should write the compareTo() method in such a way that if this( I am referring to the this keyword here) is less than the passed object then it should return negative, if greater than positive and zero if equal.

You may be wondering why I didn’t write that logic? Because last name are strings, I have called the [compareTo() method of string class](https://beginnersbook.com/2013/12/java-string-compareto-method-example/), which does exactly the same.

However if the things we are comparing are of other type such as int then you can write the logic like this:  
Let’s say object of Employee class is (empId, empName, empAge) & we want to sort the objects by empAge.

public int compareTo(Employee e){

if(this.empAge==e.empAge)

return 0;

else if(this.empAge>e.empAge)

return 1;

else

return -1;

}

**or**

public int compareTo(Employee e){

return this.empAge > e.empAge ? 1 : this.empAge < e.empAge ? -1 : 0;

}

we have seen how to sort objects of a custom class using Comparable interface. By using Comparable we can sort the objects based on any data member.but **what if we want to have multiple sort choices and we can sort objects based on any choice**, this can be done using Comparator interface, we can create as many Comparator as we want and then we can call Collections.sort on one or more Comparator like this:

//Sorting arraylist al by Author Age

Collections.sort(al, new AuthorAgeComparator());

//Sorting arraylist al by Book Name

Collections.sort(al, new BookNameComparator());

 To call the Collections.sort method like this, we must first need to write these Comparator classes AuthorAgeComparator and BookNameComparator, along with Author class and the main class.

import java.util.\*;

class AuthorAgeComparator implements Comparator<Author>{

public int compare(Author a1,Author a2){

if(a1.auAge==a2.auAge)

return 0;

else if(a1.auAge>a2.auAge)

return 1;

else

return -1;

}

}

**BookNameComparator.java**

import java.util.\*;

public class BookNameComparator implements Comparator<Author>{

public int compare(Author a1,Author a2){

return a1.bookName.compareTo(a2.bookName);

}

}

/\*Sorting using AuthorAgeComparator\*/

Collections.sort(al, new AuthorAgeComparator());

# Comparator Interface in Java

**Comparable interface** is mainly used to sort the arrays (or lists) of **custom objects**.  
Lists (and arrays) of objects that implement Comparable interface can be sorted automatically by Collections.sort (and Arrays.sort). Before we see how to sort an objects of custom objects, lets see how we can sort elements of arrays and Wrapper classes that already implements Comparable.

## Example: Sorting arrays and Wrapper class

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Collections;

import java.util.List;

public class Demo {

public static void main(String[] args) {

/\*

\* Integer class implements Comparable

\* Interface so we can use the sort method

\*/

int[] arr = {11,55,22,0,89};

Arrays.sort(arr);

System.out.print("Sorted Int Array: ");

System.out.println(Arrays.toString(arr));

/\*

\* String class implements Comparable

\* Interface so we can use the sort method

\*/

System.out.print("Sorted String Array: ");

String[] names = {"Steve", "Ajeet", "Kyle"};

Arrays.sort(names);

System.out.println(Arrays.toString(names));

/\*

\* String class implements Comparable

\* Interface so we can use the sort method

\*/

System.out.print("Sorted List: ");

List fruits = new ArrayList();

fruits.add("Orange");

fruits.add("Banana");

fruits.add("Apple");

fruits.add("Guava");

fruits.add("Grapes");

Collections.sort(fruits);

for(String s: fruits) System.out.print(s+", ");

}

}

**Output:**

Sorted Int Array: [0, 11, 22, 55, 89]

Sorted String Array: [Ajeet, Kyle, Steve]

Sorted List: Apple, Banana, Grapes, Guava, Orange,

In the above example, you have seen that how easy it is to sort the Arrays and list of objects that implements Comparable interface, you just need to call the Collections.sort (and Arrays.sort).  
**However if you want to sort the objects of custom class then you need to implement the Comparable interface in our custom class.**

This interface has only one method which is:

public abstract int compareTo(T obj)

Since this method is abstract, you must implement this method in your class if you implement the Comparable interface.

Let’s take an example to understand this better:

## Example: Sorting Custom object by implementing Comparable interface

As you can see I have implemented the Comparable interface in my Author class because I want to sort the objects of this class. I have written the logic of sorting in the compareTo() method, you can write logic based on the requirement. I wanted to sort the author names by last name first and if the last name is same then by first name. If you want to sort by the last name only then first line inside compareTo() method is enough.

**Author class**

public class Author implements Comparable<Author> {

String firstName;

String lastName;

String bookName;

Author(String first, String last, String book){

this.firstName = first;

this.lastName = last;

this.bookName = book;

}

@Override

/\*

\* This is where we write the logic to sort. This method sort

\* automatically by the first name in case that the last name is

\* the same.

\*/

public int compareTo(Author au){

/\*

\* Sorting by last name. compareTo should return < 0 if this(keyword)

\* is supposed to be less than au, > 0 if this is supposed to be

\* greater than object au and 0 if they are supposed to be equal.

\*/

int last = this.lastName.compareTo(au.lastName);

//Sorting by first name if last name is same d

return last == 0 ? this.firstName.compareTo(au.firstName) : last;

}

}

**Sorting class: SortAuthByNames**

import java.util.ArrayList;

import java.util.Collections;

public class SortAuthByNames{

public static void main(String args[]){

// List of objects of Author class

ArrayList<Author> al=new ArrayList<Author>();

al.add(new Author("Henry","Miller", "Tropic of Cancer"));

al.add(new Author("Nalo","Hopkinson", "Brown Girl in the Ring"));

al.add(new Author("Frank","Miller", "300"));

al.add(new Author("Deborah","Hopkinson", "Sky Boys"));

al.add(new Author("George R. R.","Martin", "Song of Ice and Fire"));

/\*

\* Sorting the list using Collections.sort() method, we

\* can use this method because we have implemented the

\* Comparable interface in our user defined class Author

\*/

Collections.sort(al);

for(Author str:al){

System.out.println(str.firstName+" "+

str.lastName+" "+"Book: "+str.bookName);

}

}

}

**Output:**

Deborah Hopkinson Book: Sky Boys

Nalo Hopkinson Book: Brown Girl in the Ring

George R. R. Martin Book: A Song of Ice and Fire

Frank Miller Book: 300

Henry Miller Book: Tropic of Cancer

Note: We should write the compareTo() method in such a way that if this( I am referring to the this keyword here) is less than the passed object then it should return negative, if greater than positive and zero if equal.

You may be wondering why I didn’t write that logic? Because first name and last name are strings, I have called the [compareTo() method of string class](https://beginnersbook.com/2013/12/java-string-compareto-method-example/), which does exactly the same.

However if the things we are comparing are of other type such as int then you can write the logic like this:  
Let’s say object of Employee class is (empId, empName, empAge) and we want to sort the objects by empAge.

public int compareTo(Employee e){

if(this.empAge==e.empAge)

return 0;

else if(this.empAge>e.empAge)

return 1;

else

return -1;

}

**or**

public int compareTo(Employee e){

return this.empAge > e.empAge ? 1 : this.empAge < e.empAge ? -1 : 0;

}

we have seen how to sort objects of a custom class using Comparable interface. By using Comparable we can sort the objects based on any data member. For example, lets say we have an Author class has data members: Author name, book name and author age, now if we want to sort the objects based on any of the data member then we can use Comparable but **what if we want to have multiple sort choices and we can sort objects based on any choice**, this can be done using Comparator interface, we can create as many Comparator as we want and then we can call Collections.sort on one or more Comparator like this:

//Sorting arraylist al by Author Age

Collections.sort(al, new AuthorAgeComparator());

//Sorting arraylist al by Book Name

Collections.sort(al, new BookNameComparator());

So how does it work? To call the Collections.sort method like this, we must first need to write these Comparator classes AuthorAgeComparator and BookNameComparator, along with Author class and the main class.

## Complete Comparator Example

**Author.java**

public class Author implements Comparable<Author> {

String firstName;

String bookName;

int auAge;

Author(String first, String book, int age){

this.firstName = first;

this.bookName = book;

this.auAge = age;

}

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getBookName() {

return bookName;

}

public void setBookName(String bookName) {

this.bookName = bookName;

}

public int getAuAge() {

return auAge;

}

public void setAuAge(int auAge) {

this.auAge = auAge;

}

@Override

/\*

\* When we only use Comparable, this is where we write sorting

\* logic. This method is called when we implement the Comparable

\* interface in our class and call Collections.sort()

\*/

public int compareTo(Author au){

return this.firstName.compareTo(au.firstName);

}

}

**AuthorAgeComparator.java**

import java.util.\*;

class AuthorAgeComparator implements Comparator<Author>{

public int compare(Author a1,Author a2){

if(a1.auAge==a2.auAge)

return 0;

else if(a1.auAge>a2.auAge)

return 1;

else

return -1;

}

}

**BookNameComparator.java**

import java.util.\*;

public class BookNameComparator implements Comparator<Author>{

public int compare(Author a1,Author a2){

return a1.bookName.compareTo(a2.bookName);

}

}

**SortingPgm.java**

import java.util.ArrayList;

import java.util.Collections;

public class SortingPgm{

public static void main(String args[]){

// List of objects of Author class

ArrayList<Author> al=new ArrayList<Author>();

al.add(new Author("Henry", "Tropic of Cancer",  45));

al.add(new Author("Nalo", "Brown Girl in the Ring", 56));

al.add(new Author("Frank", "300", 65));

al.add(new Author("Deborah", "Sky Boys", 51));

al.add(new Author("George R. R.", "A Song of Ice and Fire", 62));

/\*

\* Sorting the list using Collections.sort() method, we

\* can use this method because we have implemented the

\* Comparable interface in our user defined class Author

\*/

System.out.println("Sorting by Author First Name:");

Collections.sort(al);

for(Author au: al){

System.out.println(au.getFirstName()+", "+au.getBookName()+", "+

au.getAuAge());

}

/\*Sorting using AuthorAgeComparator\*/

System.out.println("Sorting by Author Age:");

Collections.sort(al, new AuthorAgeComparator());

for(Author au: al){

System.out.println(au.getFirstName()+", "+au.getBookName()+", "+

au.getAuAge());

}

/\*Sorting using BookNameComparator\*/

System.out.println("Sorting by Book Name:");

Collections.sort(al, new BookNameComparator());

for(Author au: al){

System.out.println(au.getFirstName()+", "+au.getBookName()+", "+

au.getAuAge());

}

}

}

**Output:**

Sorting by Author First Name:

Deborah, Sky Boys, 51

Frank, 300, 65

George R. R., A Song of Ice and Fire, 62

Henry, Tropic of Cancer, 45

Nalo, Brown Girl in the Ring, 56

Sorting by Author Age:

Henry, Tropic of Cancer, 45

Deborah, Sky Boys, 51

Nalo, Brown Girl in the Ring, 56

George R. R., A Song of Ice and Fire, 62

Frank, 300, 65

Sorting by Book Name:

Frank, 300, 65

George R. R., A Song of Ice and Fire, 62

Nalo, Brown Girl in the Ring, 56

Deborah, Sky Boys, 51

Henry, Tropic of Cancer, 45