# 1. Lezione #34

## 1.1. JDBC Esempio Connessione statica con getione eccezione

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class ConnectDB {

        static private final String jdbcUrl =
"jdbc:mysql://localhost/iscritticorsi?user=root";
        static private Connection connection = null;

        public static Connection getConnection() {

                try {
                        if (connection == null || connection.isClosed()) {
                                connection = DriverManager.getConnection(jdbcUrl);
                        }
                        return connection;

                } catch (SQLException e) {

                        throw new RuntimeException("Cannot get connection " +
jdbcUrl, e);
                }
        }

}
```

## 1.2. File IO/NIO Leggere Scrivere - 4 scorrere directories

```java
import java.nio.file.FileSystems;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardWatchEventKinds;
import java.nio.file.WatchEvent;
import java.nio.file.WatchKey;
import java.nio.file.WatchService;
import java.util.HashMap;
import java.util.Map;

public class Main {

        public static void main(String[] args) {

                try (WatchService service =
FileSystems.getDefault().newWatchService()) {
                        Map<WatchKey, Path> keyMap = new HashMap<>();
                        Path path = Paths.get("files");
```

```
                            keyMap.put(path.register(service,
                                            StandardWatchEventKinds.ENTRY_CREATE,
                                            StandardWatchEventKinds.ENTRY_DELETE,
                                            StandardWatchEventKinds.ENTRY_MODIFY),
                                            path);

                            WatchKey watchKey;

                            do {
                                    watchKey = service.take();
                                    Path eventDir = keyMap.get(watchKey);

                                    for (WatchEvent<?> event : watchKey.pollEvents())
{
                                            WatchEvent.Kind<?> kind = event.kind();
                                            Path eventPath = (Path)event.context();
                                            System.out.println(eventDir + ": " + kind
+ ": " + eventPath);
                                    }

                            } while (watchKey.reset());
                    } catch (Exception e) {
                            // TODO: handle exception
                    }


        }
}
```

## 1.3. File IO/NIO Leggere Scrivere - 6 CharactersStream

```


import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class Main {

        public static void main(String[] args) {

                try (
                                BufferedReader in = new BufferedReader(new
FileReader("hamlet.xml"));
                                BufferedWriter out = new BufferedWriter(new
FileWriter("newfile.txt"));
                                ) {
                        int c;
                        while ((c = in.read()) != -1) {
                                out.write(c);
                        }
                        System.out.println("All done!");
                } catch (FileNotFoundException e) {
                        e.printStackTrace();
                } catch (IOException e) {
                        e.printStackTrace();
                }

        }
}
```

## 1.4. File IO/NIO Leggere Scrivere - 8 File da Url

```


import java.io.BufferedInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.URL;

public class Main {

    private static final String FLOWERS_FEED =
            "http://services.hanselandpetal.com/feeds/flowers.xml";

    public static void main(String[] args) throws IOException {

        InputStream stream = null;
        BufferedInputStream buf = null;

        try {
            URL url = new URL(FLOWERS_FEED);
            stream = url.openStream();
            buf = new BufferedInputStream(stream);

            StringBuilder sb = new StringBuilder();

            while (true) {
                int data = buf.read();

                if (data == -1) {
                    break;
                } else {
                    sb.append((char)data);
                }
            }

            System.out.println(sb);
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            stream.close();;
            buf.close();
        }

    }

}
```

## 1.5. File IO/NIO Leggere Scrivere - 9 ByteStream

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

public class Main {

        public static void main(String[] args) {

                try (
                                FileInputStream in = new
FileInputStream("flower.jpg");
                                FileOutputStream out = new
FileOutputStream("newflower.jpg");
                                ) {
                        int c;
                        while ((c = in.read()) != -1) {
                                out.write(c);
                        }
                } catch (FileNotFoundException e) {
                        e.printStackTrace();
                } catch (IOException e) {
                        e.printStackTrace();
                }

        }
}
```

## 1.6. Java e Javascript Hello World

```
import javax.script.ScriptEngine;
import javax.script.ScriptEngineManager;
import javax.script.ScriptException;
```

```
public class Main {

        public static void main(String[] args) {

                ScriptEngineManager manager = new ScriptEngineManager();
                ScriptEngine engine = manager.getEngineByName("nashorn");

                String script = "var welcome = 'Hello'; "
                                + "welcome += ', David'; "
                                + "welcome;";

                String result;
                try {
                        result = (String)engine.eval(script);
                        System.out.println(result);
                } catch (ScriptException e) {
                        System.out.println("There was a JavaScript error");
                        e.printStackTrace();
                }

        }

}
```

## 1.7. esercizioJavaFX Anagrammi

## 1.8. Buit-in Interfaces: Comparator

```
package org.example.java8;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

public class UseComparator {

        public static void main(String args[]){

                List<String> strings = new ArrayList<String>();
                strings.add("AAA");
                strings.add("bbb");
                strings.add("CCC");
                strings.add("ddd");
                strings.add("EEE");

                //Simple case-sensitive sort operation
                Collections.sort(strings);
                System.out.println("Simple sort");
                for(String str: strings){
                        System.out.println(str);
                }

                //Case-insensitive sort with an anonymous class
                Comparator<String> comp = (str1, str2) -> {
                        return str1.compareToIgnoreCase(str2);
                };
                Collections.sort(strings, comp);
                System.out.println("Sort with comparator");
                for(String str: strings){
                        System.out.println(str);
                }

        }

}
```

## 1.9. Buit-in Interfaces: Runnable

```
package org.example.java8;

public class UseRunnable {

        public static void main(String[] args) {

//              Runnable r1 = new Runnable() {
//
//                      @Override
//                      public void run() {
//                              System.out.println("Running Thread 1");
//                      }
//              };
//
//              Runnable r2 = new Runnable() {
//
//                      @Override
//                      public void run() {
//                              System.out.println("Running Thread 2");
//                      }
//              };

                Runnable r1 = () -> {
                        try {
                                Thread.sleep(1000);
                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        }
                        System.out.println("Running Thread 1");
                };
                Runnable r2 = () -> System.out.println("Running Thread 2");
                new Thread(r1).start();
                new Thread(r2).start();

        }

}

```

## 1.10. Default methods nelle interfacce

```
package org.example.java8.interfaces;

public interface PersonInterface {

        String getName();
        void setName(String name);
        int getAge();
        void setAge(int age);

        default String getPersonInfo() {
                return getName() + " (" + getAge() + ")";
        }

}

```

## 1.11. Person implements PersonInterface

```
package org.example.java8.model;

import org.example.java8.interfaces.PersonInterface;

public class Person implements PersonInterface {
        private String name;
        private int age;

        public Person(String name, int age) {
                this.name = name;
                this.age = age;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
        public int getAge() {
                return age;
        }
        public void setAge(int age) {
                this.age = age;
        }

}
```

## 1.12. UseDefaultMethod

```
package org.example.java8;

import java.util.ArrayList;
import java.util.List;
import java.util.function.Predicate;

import org.example.java8.model.Person;

public class UseDefaultMethod {
        public static void main(String args[]){

                List<Person> people = new ArrayList<>();

                people.add(new Person("Joe", 48));
                people.add(new Person("Mary", 30));
                people.add(new Person("Mike", 73));

                Predicate<Person> pred = (p) -> p.getAge() > 65;

                displayPeople(people, pred);

        }

        private static void displayPeople(List<Person> people,
                        Predicate<Person> pred) {
                System.out.println("Selected:");
                people.forEach(p -> {
                        if (pred.test(p))
                        {
                                System.out.println(p.getPersonInfo());
                        }
                });
        }

}
```

## 1.13. Filter Collections

```
package org.example.java8.model;
```

```java
public class Person {
        private String name;
        private int age;

        public Person(String name, int age) {
                this.name = name;
                this.age = age;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
        public int getAge() {
                return age;
        }
        public void setAge(int age) {
                this.age = age;
        }

        @Override
        public String toString() {
                return this.name + " (" + this.age + ")";
        }

}
```

## 1.14. PredicateInnerClass

```java
package org.example.java8;

import java.util.ArrayList;
import java.util.List;
import java.util.function.Predicate;

import org.example.java8.model.Person;

public class PredicateInnerClass {
        public static void main(String args[]){

                List<Person> people = new ArrayList<>();

                people.add(new Person("Joe", 48));
                people.add(new Person("Mary", 30));
                people.add(new Person("Mike", 73));

                Predicate<Person> pred = new Predicate<Person>() {

                        @Override
                        public boolean test(Person p) {
                                return (p.getAge() >= 65);
                        }
                };

                for (Person person : people) {
                        if (pred.test(person)) {
                                System.out.println(person.toString());
                        }
                }

        }
}
```

## 1.15. PredicateLambda

```java
package org.example.java8;

import java.util.ArrayList;
import java.util.List;
import java.util.function.Predicate;

import org.example.java8.model.Person;

public class PredicateLambda {

        public static void main(String args[]){

                List<Person> people = new ArrayList<>();

                people.add(new Person("Joe", 48));
                people.add(new Person("Mary", 30));
                people.add(new Person("Mike", 73));

                Predicate<Person> predOlder = (p) -> p.getAge() >= 65;
                Predicate<Person> predYounger = (p) -> p.getAge() <= 40;

                displayPeople(people, predYounger);

        }

        private static void displayPeople(List<Person> people,
                        Predicate<Person> pred) {
                people.forEach( p -> {
                        if (pred.test(p)) {
                                System.out.println(p);
                        }
                });
        }
}
```

## 1.16. Functional Interfaces

```
package com.example.javase8.interfaces;

@FunctionalInterface
public interface InterfaceWithArgs {
        public void calculate(int value1, int value2);
}
```

## 1.17. SimpleInterface

```
package com.example.javase8.interfaces;

@FunctionalInterface
public interface SimpleInterface {
        public void doSomething();
}
```

## 1.18. UseInterfaceWithArgs

```
package com.example.javase8;

import com.example.javase8.interfaces.InterfaceWithArgs;

public class UseInterfaceWithArgs {

        public static void main(String[] args) {

                InterfaceWithArgs obj = (v1, v2) ->
                {
                        int result = v1 + v2;
                        System.out.println("The result is " + result);

                };
                obj.calculate(10, 5);

        }

}
```

## 1.19. UseSimpleInterface

```java
package com.example.javase8;

import com.example.javase8.interfaces.SimpleInterface;

public class UseSimpleInterface {

        public static void main(String[] args) {

                SimpleInterface obj = () -> System.out.println("Say something");
                obj.doSomething();

        }

}
```

## 1.20. Method References

```java
package org.example.java8.model;

public class Person {
        private String name;
        private int age;

        public Person(String name, int age) {
                this.name = name;
                this.age = age;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
        public int getAge() {
                return age;
        }
        public void setAge(int age) {
                this.age = age;
        }

        @Override
        public String toString() {
                return name + " (" + age + ")";
        }

        public static int compareAges(Person p1, Person p2) {
                Integer age1 = p1.getAge();
                return age1.compareTo(p2.getAge());
        }

}
```

## 1.21. InstanceMethodReference

```
package org.example.java8;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

import org.example.java8.model.Person;

public class InstanceMethodReference {

        public static void main(String args[]){
                InstanceMethodReference mainClass = new InstanceMethodReference();
                mainClass.sortData();
        }

        public void sortData() {

                List<Person> people = new ArrayList<>();

                people.add(new Person("Joe", 48));
                people.add(new Person("Mary", 30));
                people.add(new Person("Mike", 73));

                Collections.sort(people, this :: compareAges);
                people.forEach(p -> System.out.println(p) );
        }

        public int compareAges(Person p1, Person p2) {
                Integer age1 = p1.getAge();
                return age1.compareTo(p2.getAge());
        }

}
```

## 1.22. StaticMethodReference

```
package org.example.java8;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

import org.example.java8.model.Person;

public class StaticMethodReference {
        public static void main(String args[]){

                List<Person> people = new ArrayList<>();

                people.add(new Person("Joe", 48));
                people.add(new Person("Mary", 30));
                people.add(new Person("Mike", 73));

                Collections.sort(people, Person :: compareAges);
                people.forEach(p -> System.out.println(p) );
        }

}
```

## 1.23. StaticMethod

```
package org.example.java8.interfaces;

import org.example.java8.model.Person;

public interface PersonInterface {

        String getName();
        void setName(String name);
        int getAge();
        void setAge(int age);

        static String getPersonInfo(Person p) {
                return p.getName() + " (" + p.getAge() + ")";
        }
}
```

## 1.24. Person implements PersonInterface

```
package org.example.java8.model;

import org.example.java8.interfaces.PersonInterface;

public class Person implements PersonInterface {
        private String name;
        private int age;
        public Person(String name, int age) {
                this.name = name;
                this.age = age;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
        public int getAge() {
                return age;
        }
        public void setAge(int age) {
                this.age = age;
        }

}
```

## 1.25. UseStaticMethod

```
package org.example.java8;

import java.util.ArrayList;
import java.util.List;
import java.util.function.Predicate;

import org.example.java8.interfaces.PersonInterface;
import org.example.java8.model.Person;

public class UseStaticMethod {
        public static void main(String args[]){

                List<Person> people = new ArrayList<>();

                people.add(new Person("Joe", 48));
                people.add(new Person("Mary", 30));
                people.add(new Person("Mike", 73));

                Predicate<Person> pred = (p) -> p.getAge() > 65;

                displayPeople(people, pred);

        }

        private static void displayPeople(List<Person> people,
                        Predicate<Person> pred) {
                System.out.println("Selected:");
                people.forEach(p -> {
                        if (pred.test(p))
```

```
                                {
                                        String info = PersonInterface.getPersonInfo(p);
                                        System.out.println(info);
                                }
                        });
                }

        }
```

## 1.26. Traverse Collection con Comparator

```
package org.example.java8;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.Iterator;
import java.util.List;

public class Main {
        public static void main(String args[]){

                List<String> strings = new ArrayList<String>();
                strings.add("AAA");
                strings.add("bbb");
                strings.add("CCC");
                strings.add("ddd");
                strings.add("EEE");

                Collections.sort(strings);
                System.out.println("Simple sort");

                // Traverse with for:each
//              for(String str: strings){
//                      System.out.println(str);
//              }

                strings.forEach(str -> System.out.println(str));

                Comparator<String> comp = (str1, str2) ->
                {
                        return str1.compareToIgnoreCase(str2);
                };
                Collections.sort(strings, comp);

                System.out.println("Sort with comparator");

                //Traverse with iterator
//              Iterator<String> i = strings.iterator();
//              while (i.hasNext()) {
//                      System.out.println(i.next());
//              }

                strings.forEach(str -> System.out.println(str));

        }
}
```

## 1.27. Traversing Stream

```
package org.example.java8.model;

public class Person {
        private String name;
        private int age;

        public Person(String name, int age) {
                this.name = name;
                this.age = age;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
        public int getAge() {
                return age;
        }
        public void setAge(int age) {
                this.age = age;
        }

}
```

## 1.28. ParallelStream

```
package org.example.java8;

import java.util.ArrayList;
import java.util.List;
import java.util.function.Predicate;

import org.example.java8.model.Person;

public class ParallelStream {
        public static void main(String args[]){
                List<Person> people = new ArrayList<>();

                people.add(new Person("Joe", 48));
                people.add(new Person("Mary", 30));
                people.add(new Person("Mike", 73));

                Predicate<Person> pred = (p) -> p.getAge() > 65;

                displayPeople(people, pred);

        }

        private static void displayPeople(List<Person> people,
                        Predicate<Person> pred) {
                System.out.println("Selected:");

                people.stream()
                        .parallel()
                        .filter(pred)
                        .forEach(p -> System.out.println(p.getName()));

        }

}
```

## 1.29. SequentialStream

```
package org.example.java8;

import java.util.ArrayList;
import java.util.List;
import java.util.function.Predicate;

import org.example.java8.model.Person;

public class SequentialStream {
        public static void main(String args[]){

                List<Person> people = new ArrayList<>();

                people.add(new Person("Joe", 48));
                people.add(new Person("Mary", 30));
                people.add(new Person("Mike", 73));

                Predicate<Person> pred = (p) -> p.getAge() > 65;

                displayPeople(people, pred);

        }

        private static void displayPeople(List<Person> people,
                        Predicate<Person> pred) {
                System.out.println("Selected:");
//              people.forEach(p -> {
//                      if (pred.test(p))
//                      {
//                              System.out.println(p.getName());
//                      }
//              });

                people.stream()
                        .filter(pred)
                        .forEach(p -> System.out.println(p.getName()));

        }

}
```

## 1.30. Locale

```
 package CurrentLocale;
 import java.util.*;

 public class Current {


public static void main(String args[])

{

Locale lc = Locale.getDefault();


System.out.println(lc.getDisplayCountry());

System.out.println(lc.getDisplayLanguage());


System.out.println("\n");


System.out.println(lc.getCountry());

System.out.println(lc.getLanguage());


System.out.println("\n");


System.out.println(System.getProperty("user.country"));

System.out.println(System.getProperty("user.language"));
}

 }
```

## 1.31. HashMap1.java

```
 package JavaHashMap;
 import java.util.*;

 public class HashMap1 {


public static void main(String args[])

{

Map<String, String> mp = new HashMap<String, String>();


mp.put("1", "Monday");

mp.put("2", "Tuesday");

mp.put("3", "Wednesday");

mp.put("4", "Thursday");

mp.put("5", "Friday");

mp.put("6", "Saturday");

mp.put("7", "Sunday");


Iterator<Entry<String,String>> it = mp.entrySet().iterator();


while(it.hasNext())

{

Map.Entry<String,String> entry = (Map.Entry<String,String>)it.next();

System.out.println("Key: " + entry.getKey() + "Value is: " + entry.getValue());

}

}
```

```
 }
```

## 1.32. SortHashMapByKeys

```
 package SortHashMapByKeys;
 import java.util.HashMap;

 public class SortHashMap {


public static void main(String args[])

{

HashMap<String,Integer> mp = new HashMap();


mp.put("g Saturday", 6);
mp.put("e Thursday", 4);
mp.put("b Tuesday", 2);
mp.put("f Friday", 5);
mp.put("h Sunday", 7);
mp.put("c Wednesday", 3);
mp.put("a Monday", 1);


System.out.println("Maps before sorting: ");

Set st = mp.entrySet();

Iterator it = st.iterator();

while(it.hasNext())

{

Map.Entry mpen = (Map.Entry)it.next();

System.out.println(mpen.getKey() + " : " + mpen.getValue());

}


System.out.println("\n");


Map<String,Integer> mapsi = new TreeMap(mp);
System.out.println("Maps after sorting: ");

Set st1 = mapsi.entrySet();

Iterator it1 = st1.iterator();

    while(it1.hasNext())

        {

        Map.Entry mpen1 = (Map.Entry)it1.next();

        System.out.println(mpen1.getKey() + " : " + mpen1.getValue());

    }

}


 }
```

## 1.33. SortHashMap.java

```


 package SortHashMapByValues;
 import java.util.*;

 public class SortHashMap {


public static void main(String args[])

{

HashMap<Integer,String> hm = new HashMap<Integer,String>();

hm.put(7, "Sunday");
hm.put(4, "Thursday");
hm.put(2, "Tuesday");
hm.put(5, "Friday");
hm.put(1, "Monday");
hm.put(6, "Saturday");
hm.put(3, "Wednesday");

System.out.println("Map before sorting: ");

Set st = hm.entrySet();

Iterator it = st.iterator();
while(it.hasNext())

{

Map.Entry mpen = (Map.Entry)it.next();

System.out.println(mpen.getKey() + " : " + mpen.getValue());

}


Map<Integer,String> mp = sortByValues(hm);

System.out.println("Map after sorting: ");

Set st1 = mp.entrySet();

Iterator it1 = st1.iterator();


while(it1.hasNext())

{

Map.Entry mpen1 = (Map.Entry)it1.next();

System.out.println(mpen1.getKey() + " : " + mpen1.getValue());

}

}
```

## 1.34. SortHashMapByKeys

```


public static c(HashMap mp)

{

List lt = new LinkedList(mp.entrySet());

Collections.sort(lt, new Comparator()

{


public int compare(Object o1, Object o2) {

return((Comparable)((Map.Entry)(o1)).getValue()).compareTo(((Map.Entry)
(o2)).getValue());
}

});

HashMap sorted = new LinkedHashMap();

for(Iterator it2 = lt.iterator(); it2.hasNext();)

{

Map.Entry ent = (Map.Entry)it2.next();

sorted.put(ent.getKey(), ent.getValue());

}

return sorted;

}
}
```

## 1.35. Employee implements Comparable

```
 import java.util.*;

 public class Employee implements Comparable<Employee>{


private String name;

private String occupation;

private int salary;


public Employee(String firstname, String job, int value)

{

super();

this.name = firstname;

this.occupation = job;

this.salary = value;

}

```

```
public String getName() {

return name;

}


public void setName(String name) {

this.name = name;

}


public String getOccupation() {

return occupation;

}


public void setOccupation(String occupation) {

this.occupation = occupation;

}


public int getSalary() {

return salary;

}


public void setSalary(int salary) {

this.salary = salary;

}


public String toString()

{

return "Name of employee is: " + name + " Occupation is: " + occupation + " Salary
is:
" + salary;

}


@Override
```

```
public int compareTo(Employee comparemydata) {


int compareValue = ((Employee)comparemydata).getSalary();


return this.salary - compareValue;

}


public static void main(String args[])

{
Employee employee1 = new Employee("Hello1","Programmer1",1000);
Employee employee2 = new Employee("Hello2","Programmer2",7000);
Employee employee3 = new Employee("Hello3","Programme3",6000);
Employee employee4 = new Employee("Hello4","Programmer4",4000);
Employee employee5 = new Employee("Hello5","Programmer5",8000);

List<Employee> employee = new ArrayList();
employee.add(employee1);
employee.add(employee2);
employee.add(employee3);
employee.add(employee4);
employee.add(employee5);
System.out.println("Objects before sorting: ");
for(Employee empl:employee)
{
System.out.println(empl);
}
System.out.println("\n");
Collections.sort(employee);
System.out.println("Objects after sorting: ");
for(Employee empl:employee)
{
System.out.println(empl);
}

 }

}
```

## 1.36. sortObjects implements Comparator

```
import java.util.*;
class sortObjects implements Comparator<Employee>
{
```

```
@Override

public int compare(Employee employee1, Employee employee2) {


return (employee1.getSalary() - employee2.getSalary());

}


public static void main(String args[])

{
Employee employee1 = new Employee("Hello1","Programmer1",1000);
Employee employee2 = new Employee("Hello2","Programmer2",7000);
Employee employee3 = new Employee("Hello3","Programme3",6000);
Employee employee4 = new Employee("Hello4","Programmer4",4000);
Employee employee5 = new Employee("Hello5","Programmer5",8000);

List<Employee> employeelist = new ArrayList();

employeelist.add(employee1);
employeelist.add(employee2);
employeelist.add(employee3);
employeelist.add(employee4);
employeelist.add(employee5);
System.out.println("Objects before sorting: ");
for(Employee employee : employeelist)
{
    System.out.println(employee);
}
System.out.println("\n");
Collections.sort(employeelist, new sortObjects());
System.out.println("Objects after sorting: ");
for(Employee employee : employeelist)
{
    System.out.println(employee);
}

}

 }
```

## 1.37. FirstNonRepeatedCharacter

```
 import java.util.*;

 public class NonRepeated {


```

```
public static void main(String args[])

{

Map<Character, Integer> chmap = new HashMap();
Scanner sc = new Scanner(System.in);
System.out.println("Please insert only string value: ");
String Str1 = sc.nextLine();
for(int x=0; x<Str1.length(); x++)
{
    char ch = Str1.charAt(x);
if(chmap.containsKey(ch))
{
    chmap.put(ch, chmap.get(ch)+1);
}
else
{
    chmap.put(ch, 1);
}
}


for(int x=0; x<Str1.length(); x++)

{

if(chmap.get(Str1.charAt(x)) == 1)

{

System.out.println("First non-repeated character of " + Str1 + " is " +
Str1.charAt(x));

break;

}

}

}

 }
```

## 1.38.

```
```

## 1.39.

```
```

## 1.40.

```
```