



# *Programming Foundations*

## **FIT9131**

### Assignment Project Exam Help

*Documentation, identity,  
equality, more collections*

*Week 6*

# *Lecture outline*

- Class documentation
  - javadoc
- Object identity vs object equality
  - Java Strings
    - <https://powcoder.com>
- Iterators
  - Add WeChat powcoder
- Conditional operator

# Review : Class documentation

- The Java standard library documentation shows details about all classes in the library.
- This is called the *Java API (Application Programming Interface)* documentation.  
<https://powcoder.com>
- The documentation is readable in a Web browser.
- Provides an *interface* description for all library classes

# Interface vs. Implementation

The documentation includes:

- the *name* of the class;
- a general *description* of the class; Assignment Project Exam Help
- a list of *constructors* and *methods*;
- *return value types* and *parameters* for constructors and methods; https://powcoder.com Add WeChat powcoder
- a description of the *purpose* of each constructor and method.



describes the *interface* of the class

# Interface vs. Implementation

The documentation *does not* include :

- private fields (fields are typically private)
- private methods
- the bodies (source code) for each method

Add WeChat powcoder



does not describe the *implementation* of the class

# Writing class documentation

- It is important to document your code.
- Your own classes should be documented in the same way as the Java library classes  
*Assignment Project Exam Help*  
<https://powcoder.com>
- *Important* : other people should be able to use your class without reading/knowing the implementation.  
*Add WeChat powcoder*
- Make your class to be just like a 'library class'! The aim is to *make your class as re-usable as possible* – and not just for use in the current program you are writing.

# Elements of documentation

Documentation for a **class** should include at least:

- the **class name**
- a **comment** describing the overall **purpose** and **characteristics** of the class
- a **version number**
- the **authors' names**
- documentation for each **constructor** and each **method**

# Elements of documentation

The documentation for each **method** (and also each **constructor**) should include:

- the **name** of the method
- the **return type**
- the **parameter names and types**
- a description of the **purpose** and **function** of the method
- a description of each **parameter**
- a description of the **value** returned

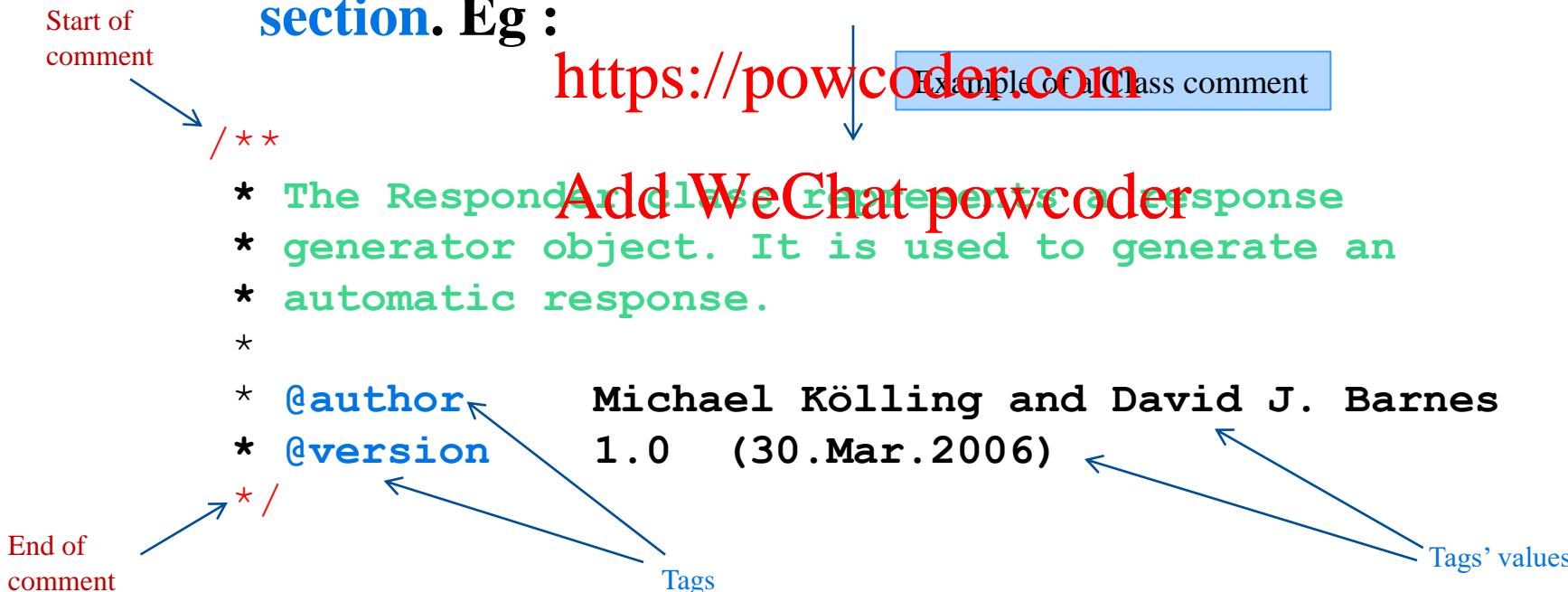
# Javadoc

*Javadoc* is a tool (a document generator) to *automate* the generation of Java documentation.

Documentation comments are enclosed by `/** ... */`

These consist of a main description followed by a "tag" section. Eg :

```
/*
 * The Response class represents a response
 * generator object. It is used to generate an
 * automatic response.
 *
 * @author Michael Kölling and David J. Barnes
 * @version 1.0 (30.Mar.2006)
 */
```



# Javadoc (contd.)

## Method comment:

Main description

Example of a Method comment

```
/**  
 * Read Assignment Project Exam Help  
 * from standard input (the text  
 * terminal), and return it as a set of words.  
 */
```

<https://powcoder.com>

Tags

```
* @param prompt A prompt to print to screen.  
* @return An ArrayList set of Strings, where each  
*         String is one of the words typed by the user
```

\*/

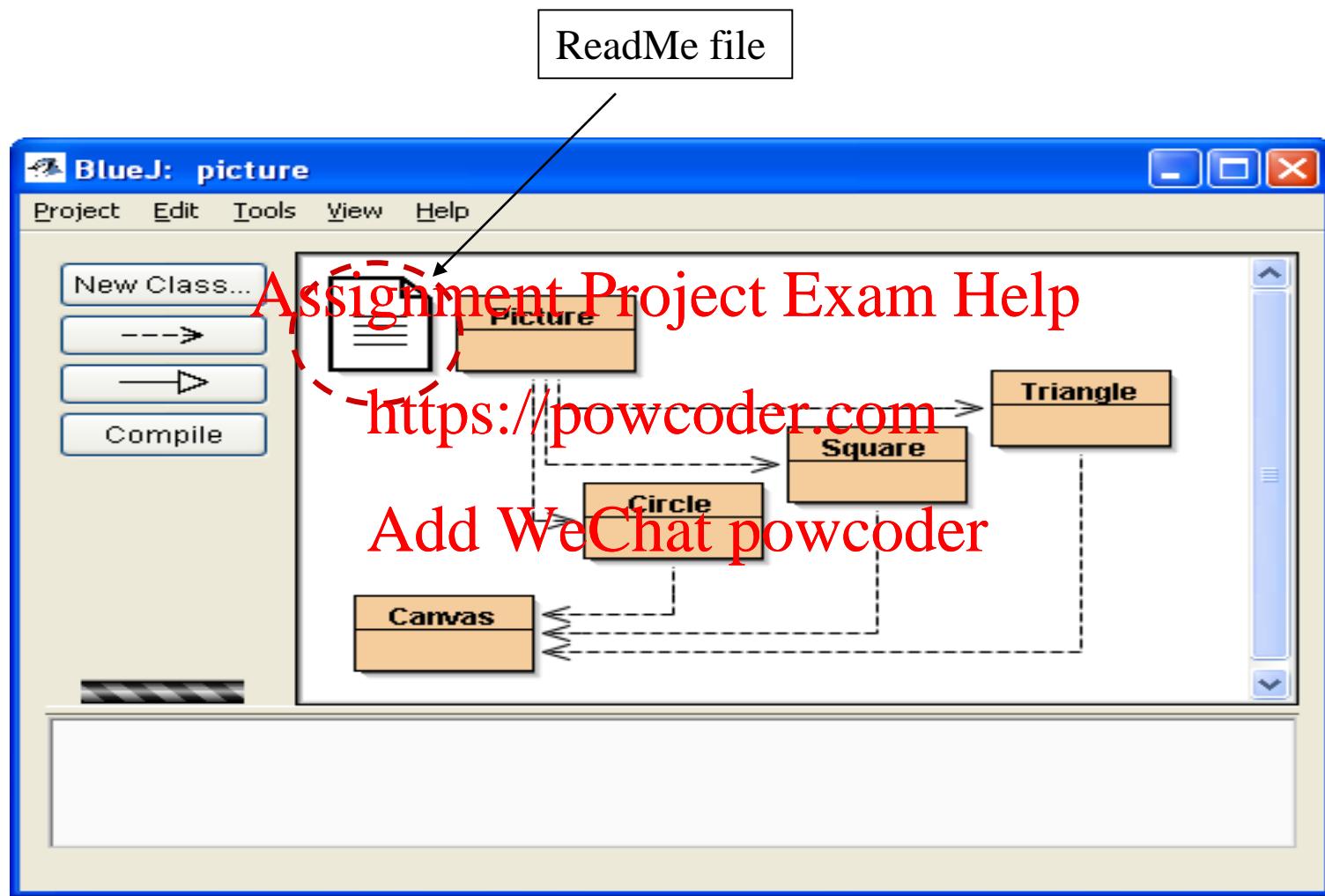
```
public ArrayList<String> getInput(String prompt)
```

```
{
```

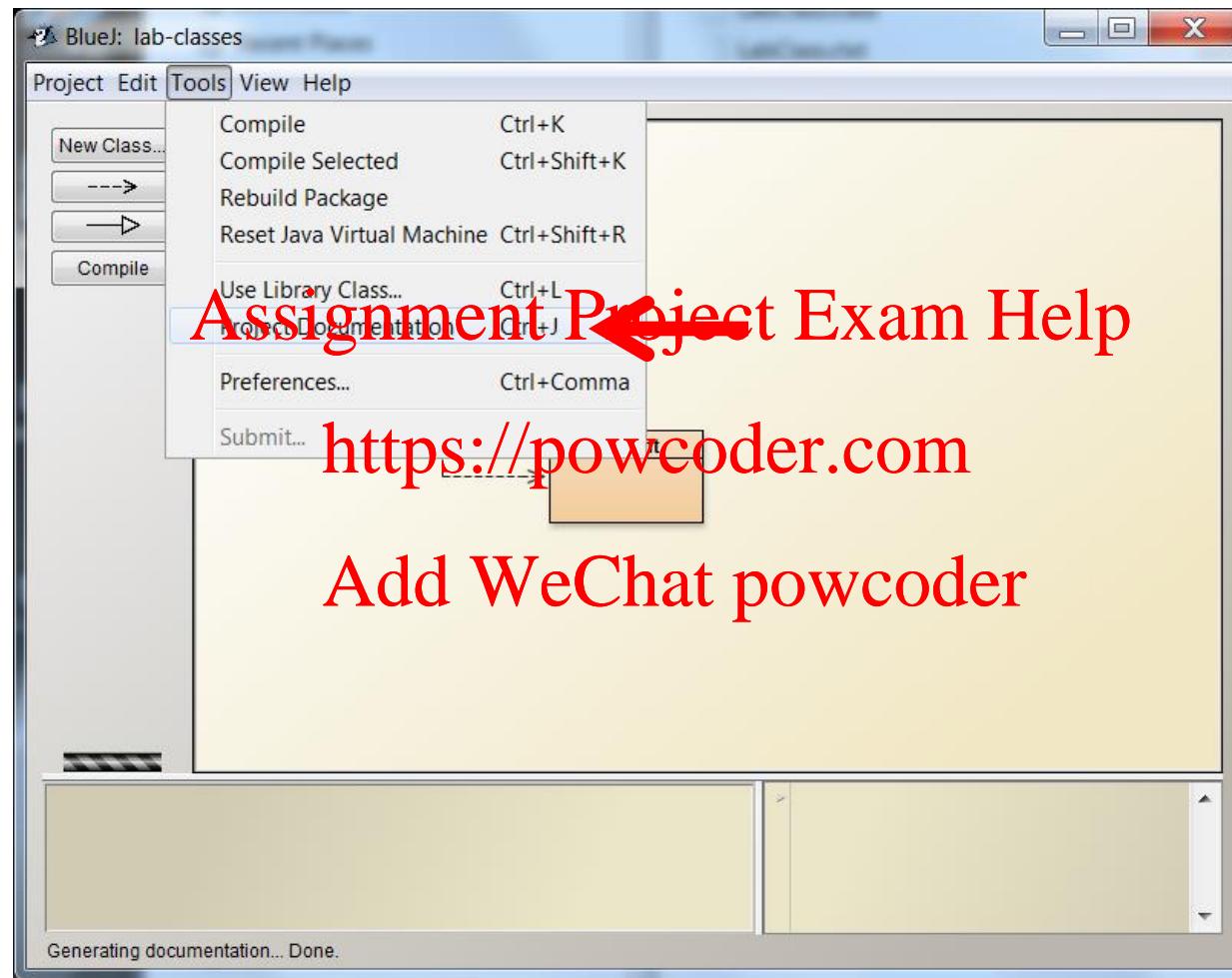
```
    ...
```

```
}
```

# Overall project details (BlueJ)



# Using BlueJ to generate documentation



# Sample Javadoc output from BlueJ

The screenshot shows a Java documentation browser window for the class `LabClass`. The browser has a header with tabs for **Package**, **Class**, **Tree**, **Index**, and **Help**. Below the tabs are links for **PREV CLASS**, **NEXT CLASS**, **SUMMARY: NESTED | FIELD**, **CONSTR**, and **METHOD**. The main content area is titled **Class LabClass**. It shows the inheritance path: `java.lang.Object` → `LabClass`. A detailed description of the `LabClass` follows:

**Version:** 2011.07.31  
**Author:** Michael Kolling and David Barnes

**Constructor Summary**

`LabClass(int maxNumberOfStudents)`  
Create a LabClass with a maximum number of enrolments.

**Method Summary**

<code>void</code>	<code>enrolStudent(Student student)</code> Add a student to this LabClass.
<code>int</code>	<code>numberOfStudents()</code> Return the number of students currently enrolled in this LabClass.
<code>void</code>	<code>printList()</code> Print out a class list with other LabClass details to the standard terminal.
<code>void</code>	<code>setInstructor(String instructorName)</code> Set the name of the instructor for this LabClass.
<code>void</code>	<code>setRoom(String roomNumber)</code> Set the room number for this LabClass.
<code>void</code>	<code>setTime(String timeAndDayString)</code> Set the time for this LabClass.

**Methods inherited from class**

`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

**Constructor Detail**

`LabClass`

A large red watermark across the center of the page reads: **Assignment Project Exam Help** and **https://powcoder.com**.

Add WeChat powcoder

# Toggling between Document/SourceCode view

```

LabClass - lab-classes
Class Edit Tools Options
Compile Undo Cut Copy Paste Find Close
Source Code Documentation
import java.util.*;

/**
 * The LabClass class represents an enrolment list for one lab class. It stores
 * the time, room and participants of the lab, as well as the instructor's name.
 *
 * @author Michael Kölbing and David Barnes
 * @version 2011.07.31
 */
public class LabClass
{
    private String instructor;
    private String room;
    private String timeAndDay;
    private ArrayList<Student> students;
    private int capacity;

    /**
     * Create a LabClass with a maximum number of enrolments. All other details
     * are set to default values.
     */
    public LabClass(int maxNumberOfStudents)
    {
        instructor = "unknown";
        room = "unknown";
        timeAndDay = "unknown";
        students = new ArrayList<Student>();
        capacity = maxNumberOfStudents;
    }

    /**
     * Add a student to this LabClass.
     */
    public void enrollStudent(Student newStudent)
    {
        if(students.size() == capacity) {
            System.out.println("The class is full, you cannot enrol.");
        }
        else {
            students.add(newStudent);
        }
    }
}

```

saved

Source Code view

LabClass - lab-classes

Class Edit Tools Options

Source Code Documentation

PREV CLASS CLASS SUMMARY NESTED FIELD METHOD

ERASE ALL NO FRAMES ALL CLASSES DETAIL FIELD CONSTR METHOD

Class LabClass

EXTENDS Object

The LabClass class represents an enrolment list for one lab class. It stores the time, room and participants of the lab, as well as the instructor's name.

Version: 2011.07.31  
Author: Michael Kölbing and David Barnes

**Constructor Summary**

- public LabClass(int maxNumberOfStudents)

**Method Summary**

- void `enrollStudent(Student newStudent)` Add a student to this LabClass.
- int `numberOfStudents()` Return the number of students currently enrolled in this LabClass.
- void `printList()` Print out a class list with other LabClass details to the standard terminal.
- void `setInstructor(String instructorName)` Set the name of the instructor for this LabClass.
- void `setRoom(String roomNumber)` Set the room number for this LabClass.
- void `setTimeOfDay(String timeAndDayString)` Set the time for this LabClass.

Methods inherited from class

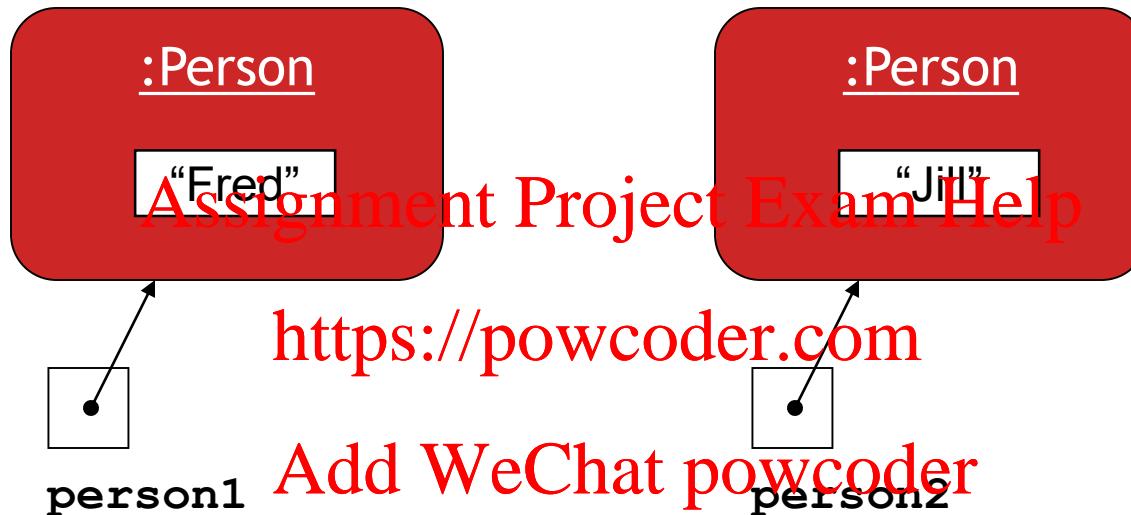
- clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Loading class interface... Done.

saved

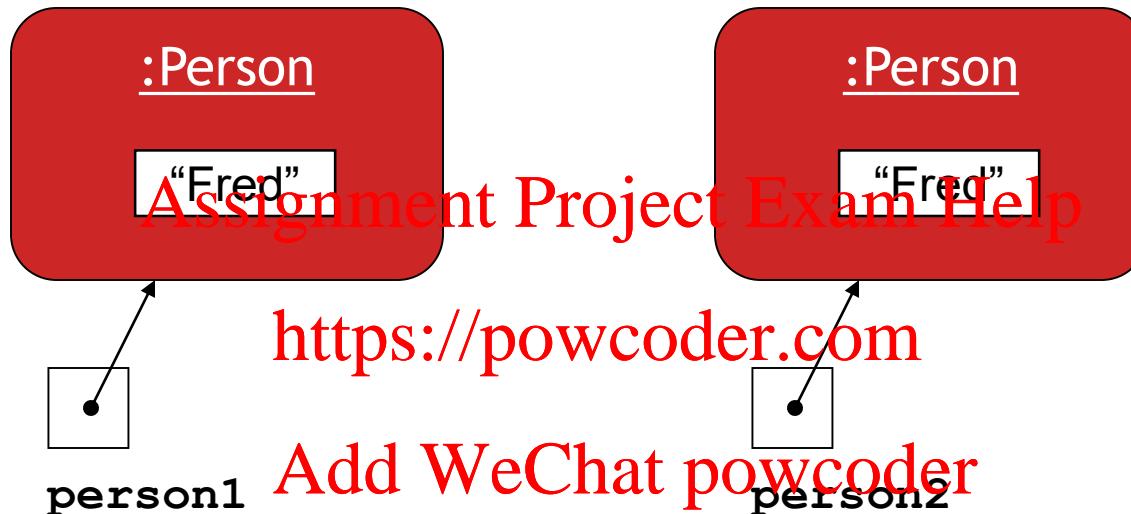
Document view

# Object : Identity vs. Equality



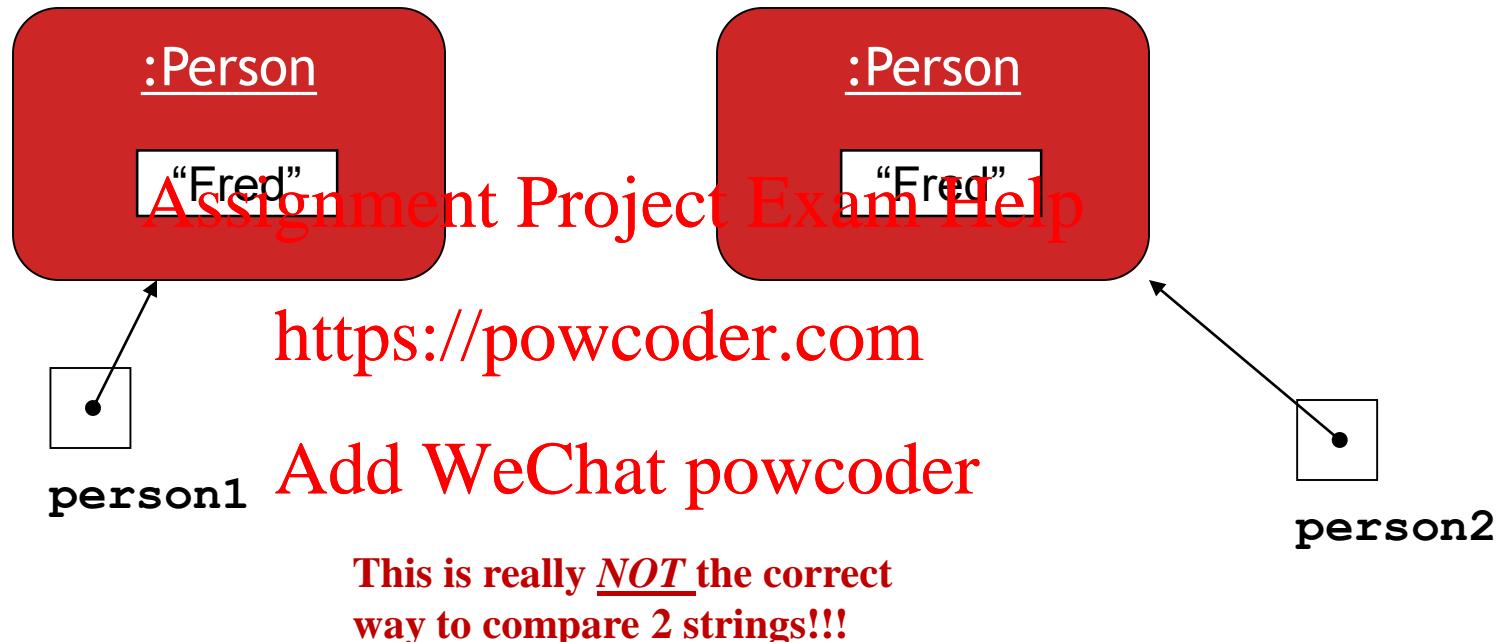
`person1 == person2 ?`

# Object : Identity vs. Equality



`person1 == person2 ?`

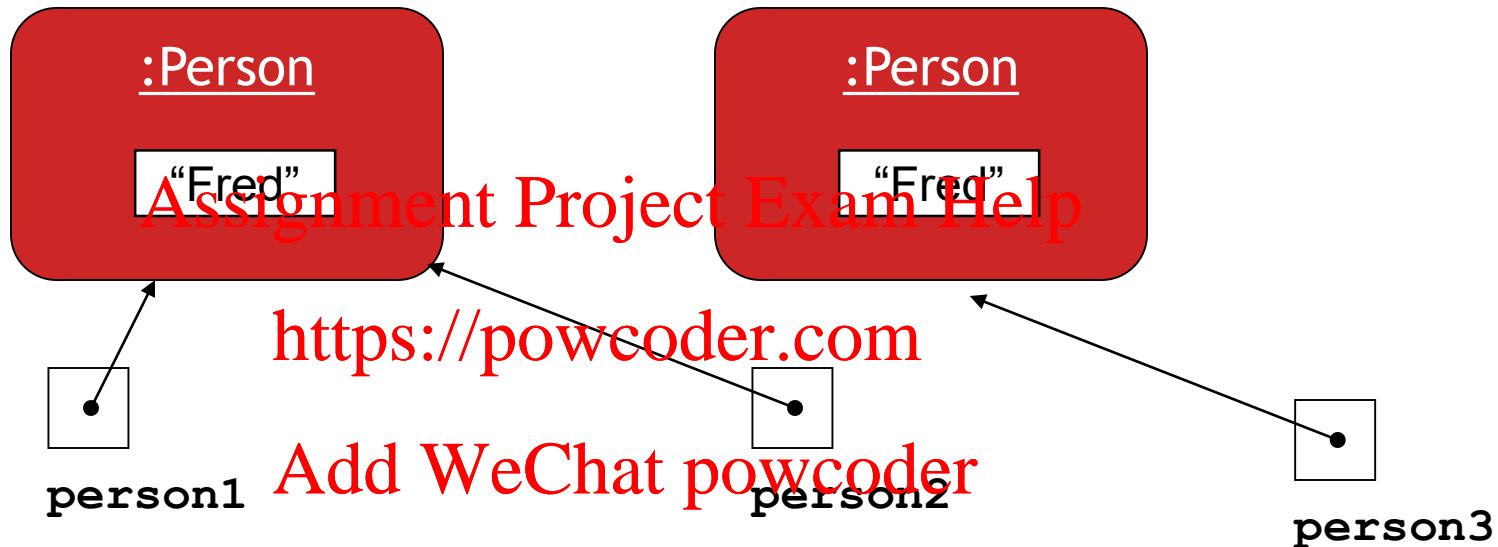
# Object : Identity vs. Equality



```
person1.getName() == person2.getName() ?
```

(assuming **Person** class has a **getName()** accessor method)

# Object : Identity vs. Equality



`person1 == person2 ?`

`person1 == person3 ?`

# Comparing two Strings

A **java String is an object**. Comparing two **Strings** for equality using the `==` operator compares the reference variables, i.e. the memory locations, for equality. The only way they will be equal is if they are the same location in memory, i.e. the same object.

Assignment Project Exam Help

If you want to compare the *content* of two **Strings** to see if they are the same, you should use the `equals` method of the **String** class.

The same rule applies to any two objects. If you write a class, and you want to compare the state (eg. the name attribute of a **Person** class) of the two objects of that class, you need to write an `equals` method for the class.

# String equality

```
if(input == "bye")  
{  
    ...  
}
```



tests identity

Assignment Project Exam Help

```
if(input.equals("bye"))  
{  
    ...  
}
```



tests equality

https://powcoder.com  
Add WeChat powcoder

Strings should always be compared using the `equals` method.  
The `String` class also has an `equalsIgnoreCase` method.

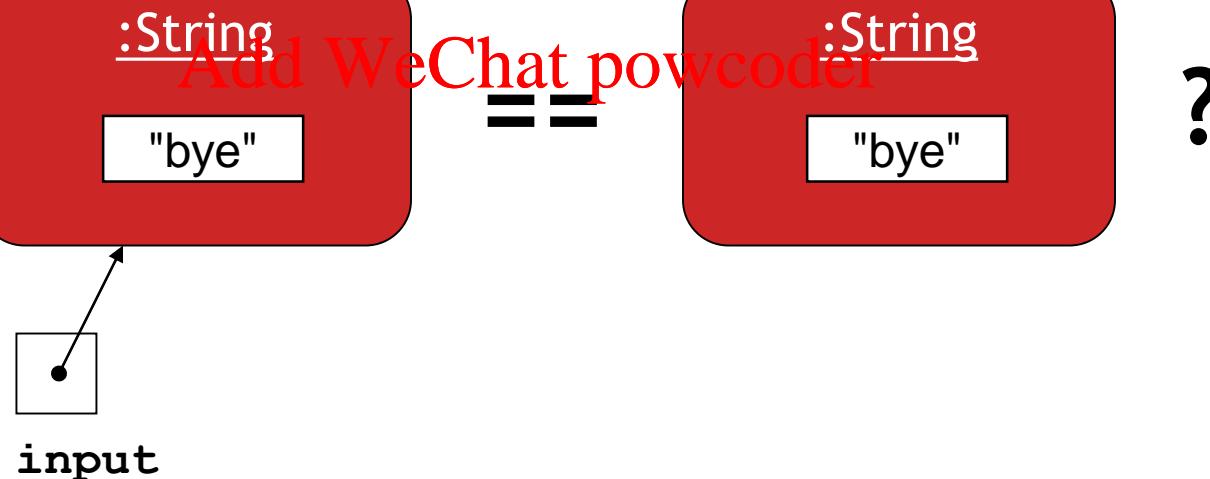
# Identity vs. equality (Strings)

This example is from the `InputReader` class in the `tech-support` project from the textbook (Chapter05)

```
String input = reader.getInput();  
if (input == "bye")  
{  
    ...  
}
```

Assignment Project Exam Help tests identity

<https://powcoder.com>



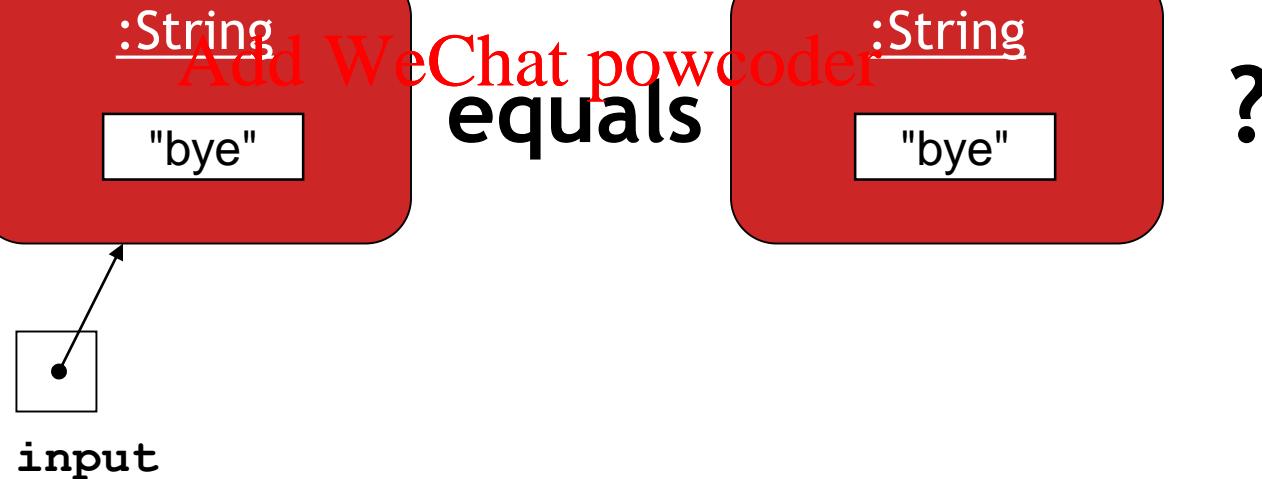
# Identity vs. equality (Strings)

```
String input = reader.getInput();  
if (input.equals("bye"))  
{  
    ...  
}
```

equals tests  
equality

Assignment Project Exam Help

<https://powcoder.com>



# Strings are immutable

Java Strings are *immutable*. A String never changes its value. A method that appears to change a String (e.g. `toUpperCase`) actually creates a new temporary String that looks like the original in upper case, but ~~the original is unchanged~~.

Eg :

```
public void changingString()  
{  
    Add WeChat powcoder  
    String greeting = "Hello there";  
  
    System.out.println(greeting.toUpperCase());  
  
    System.out.println(greeting);  
}
```

What will be displayed on the screen?

# Example: Building a string using a loop

```
// this works, but is rather inefficient
public void showLine(int numberOfStars)
{
    String line = "";
    for (int counter = 0; counter < numberOfStars;
         counter++)
        line = line + "*";
    System.out.println(line);
}
```

Assignment Project Exam Help  
<https://powcoder.com>

Add WeChat powcoder

How many temporary Strings are created here?

# Example : Changing a string (more efficient way)

```
public String changingString(String aString)
{
    StringBuffer buffer = new StringBuffer(aString);
    // modify the StringBuffer object here...
    ...
    return buffer.toString();
}
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

Strings can be converted to **StringBuffers** and vice versa. It is often more efficient to use a **StringBuffer**, especially when you are going to append things to it.

# Example: Building a string (more efficient way) again

```
public void showLineWithStringBuffer(int numberOfStars)
{
    String line = "";
    StringBuffer buffer = new StringBuffer(line);
    for (int counter = 0; counter < numberOfStars;
        counter++)
        buffer.append('*');
    System.out.println(buffer);
}
```

This is more efficient than  
the previous example. Why?

# Removing spaces from a string

This is from the `InputReader` class  
in the `tech-support` project from  
the textbook (Chapter05)

The leading and trailing spaces can be easily removed from a string:

Assignment Project Exam Help

```
String input = console.getInput();  
input = input.trim();
```

These can be ~~Added WeChat~~ merged into a single line of code :

```
String input = console.getInput().trim();
```

# *Test : Changing the case of characters in a string*

Write a line of code that will *get a line of input from the user, removes the leading and trailing spaces and also changes it to lower case*. Hint : use the ~~Assignment Project Exam Help~~ on the previous slide as a starting point.

<https://powcoder.com>

```
String input = Add WeChat powcoder
```

# Examples : Validating a String

- If a **String** is supposed to be numeric (e.g. a phone number or a postcode), you need to check that each character is a numeric character (i.e. between '0' and '9').  
**Assignment Project Exam Help**
- If a **String** is not supposed to be blank (where a blank **String** is either empty or full of space characters), you could check each character to see if they are all spaces.  
**Add WeChat powcoder**
- Unfortunately the **String** class does not provide methods to do these checks.

# Example: stringNumeric method

```
public boolean stringNumeric(String aString)
{
    if (aString.length() == 0)
        return false;
    int position = 0;
    while (position < aString.length())
    {
        char thisCharacter = aString.charAt(position);
        if ((thisCharacter < '0' || thisCharacter > '9'))
            return false;
        position++;
    }
    return true;
}
```

Assignment Project Exam Help  
<https://powcoder.com>  
Add WeChat powcoder

# Revision : loops and collections

Loop statements allow a block of statements to be repeated.

- The *while* loop allows the repetition to be controlled by a boolean expression.  
*Assignment Project Exam Help*
- The *for* loop offers an alternative to while loops when the number of repetitions is known.  
*https://powcoder.com*
- The *for-each* loop allows iteration over a whole collection.  
*Add WeChat powcoder*
- All collection classes provide a special object known as an *Iterator*, that provides sequential access to the whole collection.

# Iterator and iterator()

- Collections (eg. ArrayLists) have an `iterator()` method.
  - Calling this method returns an `Iterator` object.
- An `Iterator<E>` object has three methods:
  - `boolean hasNext()`
  - `E next()`
  - `void remove()`

# Iterating using an iterator object

An iterator object can be used to perform looping over a Collection. Eg :

- The `iterator` method of `ArrayList`, like all collections, returns an `Iterator` object.
- A `while` loop performs the iteration.
- The `iterator` object keeps track of the position in the list.

Add WeChat powcoder

To use the iterator object we need to import the Iterator class:

```
import java.util.Iterator;
```

# Using an Iterator object

`java.util.Iterator`

returns an `Iterator` object

General Syntax

```
Iterator<ElementType> it = myCollection.iterator();
while (it.hasNext())
{
    call it.next() to get the next object
    do something with that object
}
```

Assignment Project Exam Help  
https://powcoder.com

Java Example

```
public void listAllFiles()
{
    Add WeChat powcoder
    Iterator<Track> it = files.iterator();
    while (it.hasNext())
    {
        Track tk = it.next();
        System.out.println(tk.getDetails());
    }
} // MusicOrganizer example from Week 5
```

# Other Collections (non-examinable topics)

- Apart from the **ArrayList** we discussed before, Java also provides other collections. Two of the more commonly used ones are :
  - **HashMaps** : collections that contain *pairs of objects*. Pairs consist of a *key* and a *value*.
  - **HashSets** : collections that store each individual element at most once (ie. no duplicates).
- These specialised collections are often used to implement specific types of data, Eg. A **Map** can be used to implement a *Telephone Book*, where each “pair” consists of a *name* and a corresponding *tel#*.

# Collection comparisons

Collection	Strengths	Limitations	Example usages
ArrayList	A sequence of elements in order of insertion	Slow to search, slow to add/remove arbitrary elements.	List of accounts; waiting line; the lines of a file.
HashSet	A set of unique elements that is fast to search	Does not have indexes; cannot retrieve arbitrary elements from it.	Unique words in a book; lottery ticket numbers
HashMap	A group of associations between pairs of "key" and "value" objects.	Not a general-purpose collection; cannot easily map backward from a value to its key.	Word counting; phone book.

From: "Building Java Programs", by Regges & Stepp

# Conditional operator

```
public String toString()
{
    return name + " " + age + "
        + (isStudent ? "Student" : "Not a student");
}
```

*conditional  
operator*

Assignment Project Exam Help

<https://powcoder.com>

*alternative*

Add WeChat powcoder

? : is called the *conditional operator*. All three parts (**condition**, **consequent**, **alternative**) must be present. Its returned value can be used like any other returned value. How does this compare to the **if** statement?

# An equivalent IF example

```
public String toString()
{
    if (isStudent)
        return name + " " + age + " Student";
    else
        return name + " " + age + " Not a student";
}
```

*condition*

*consequent*

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

*alternative*

This does the same thing as the version using the *conditional operator*, but the code is longer.

# Common Scanner input error

Problems may occur when using `nextLine()` in conjunction with `nextInt()` from the Scanner class.

For example, when the following code was executed:

```
int age = console.nextInt();
String name = console.nextLine();
System.out.println(age + ", " + name);
```

Add WeChat powcoder  
with the following input from the keyboard:

77<enter>

Helen<enter>

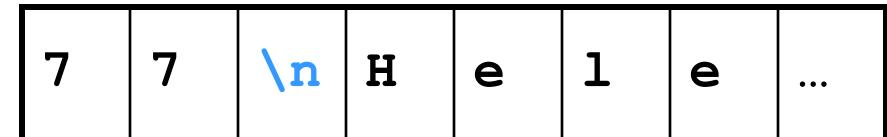
the following output will be produced:

77,

What happened to the name?

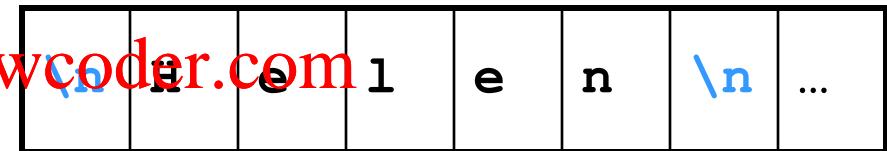
# What was the problem?

To Java, the input is a long buffer of characters:



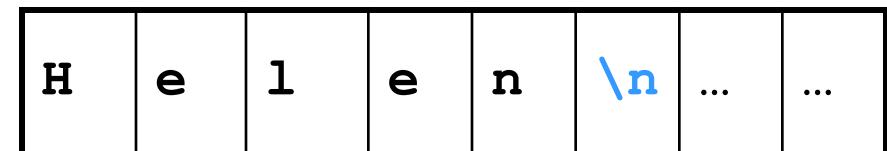
## Assignment Project Exam Help

`nextInt` removes the characters corresponding to a number, and that is all. The newline character ('\n') is not removed from the buffer.



Add WeChat powcoder  
`age = 77`

`nextLine` looks for the next newline character ('\n'), and returns everything before the first one it finds, in this example it will return an empty string.



`name = ""`

# Some possible solutions

- use `nextLine()` to get all input (this will remove the newline character properly)
  - for this approach, the inputs may need to be converted back to the correct format, eg. to int's if arithmetic operations are needed.
  - so the previous example would become :

```
String      Add WeChat nextLine()
int age = console.nextInt();
String name = console.nextLine();
System.out.println(age + ", " + name);

// obviously, this only works if we do not intend
// to perform any calculations on the age
```

# Some possible solutions

- use a dummy `nextLine()` to explicitly remove the newline character, before the next real input statement. In other words, use an extra `nextLine()` statement, but don't store the input for it.
  - so the previous example would become :

```
int age Add WeChat powcoder  
console.nextInt();  
console.nextLine(); ← a dummy getLine()  
String name = console.nextLine();  
System.out.println(age + " , " + name);  
(note that the result is not assigned to anything)
```



# The auction project

The *auction* project (Chapter04 from textbook) provides further illustration of collections and iteration... and some other concepts.

Assignment Project Exam Help

A field that has not explicitly been initialised will contain the special value `null` by default.

Add WeChat powcoder

We can test if an object variable holds the `null` variable. If it does, *it is not a valid object*, and must not be used!!

# Special value : *null*

- Used with object types.
- Used to indicate : "*no object*".
- We can ~~Assignment Project Exam Help~~ test if an object variable holds the null value, Eg :

<https://powcoder.com>

```
if (highestBid == null) . . .
Add WeChat powcoder
```

- if successful, this test indicates ‘there is no bid yet’, since the **highestBid** object does not exist.

# "Anonymous" objects

- Objects are often created and handed on elsewhere immediately:

```
Lot furtherLot = new Lot(...);  
lots.add(furtherLot);  
https://powcoder.com
```



A diagram showing the assignment of a new object to a variable. A red oval labeled "Assignment Project Exam Help" surrounds the assignment operator (=) and the object creation part (new Lot(...)). An arrow points from this oval to another red oval labeled "Add WeChat powcoder" which surrounds the entire line of code.

- If we don't really need the object **furtherLot** after this, we could have achieved the same effect using an **anonymous object**:

```
lots.add(new Lot(...));
```



A diagram showing the creation of an anonymous object. A red oval labeled "new Lot(...)" surrounds the object creation part of the code. An arrow points from this oval to a text label below it.

anonymous object (an object with no name)