UNTAR
Universitas Tarumanagara

Terakreditasi BAN PT A (Unggul)

UNTAR untuk INDONESIA

Object-based Programming

Week 3 – Library Classes and Packages

Examples of Java Library Usage

- Creating a GUI
- Sorting data
- Using network connections
- Playing sound
- Connecting to a database server

Using library classes

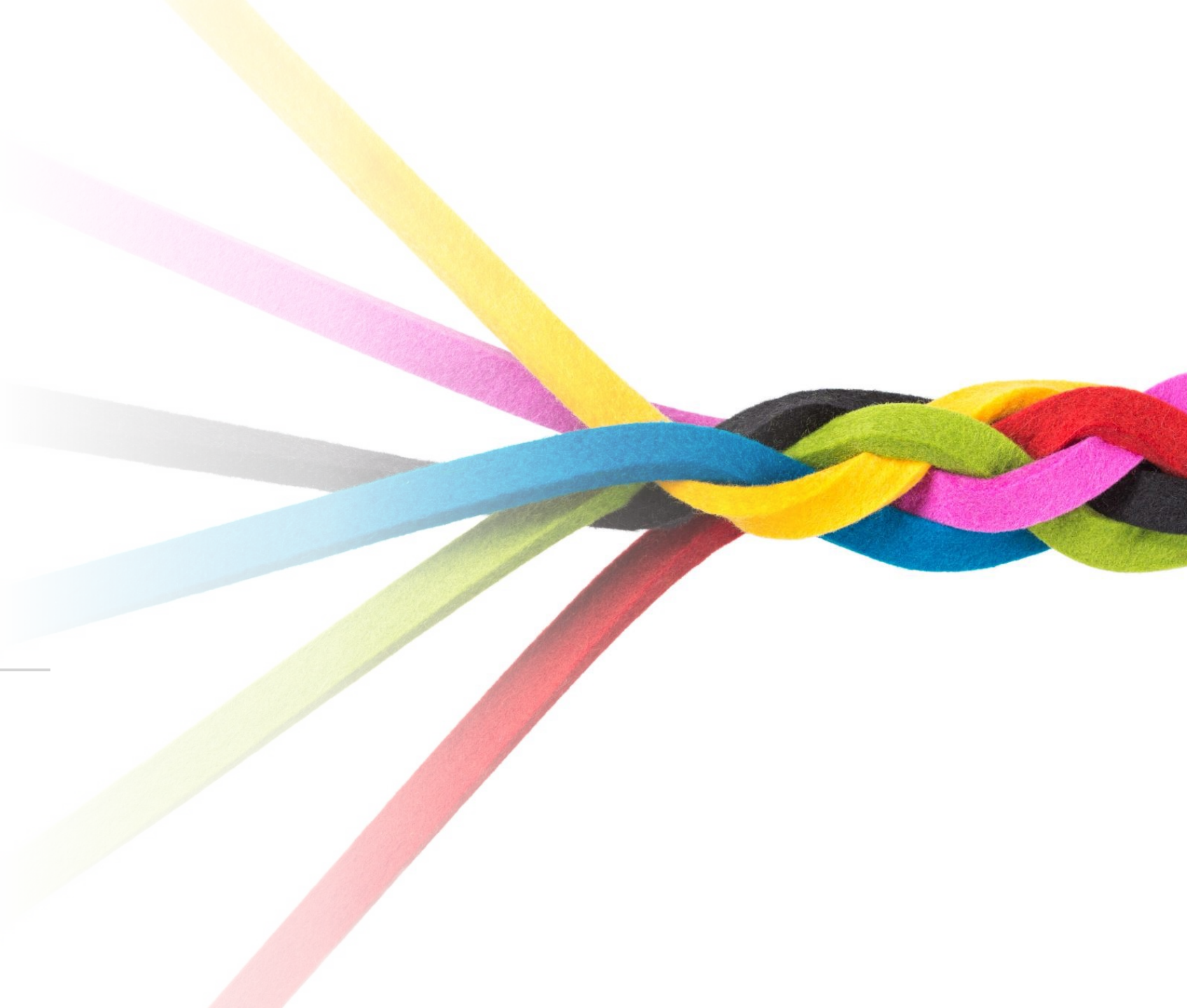
```
import java.util.Random;
// imports the random class from the java.util package

import java.util.*;
// imports all classes from the java.util package
```

- `java.lang` package is imported by default by the Java compiler

- Common usage of this package, for example `java.lang.String`

The `String` class



```
// Separate declaration and initialisation
String s1;
s1 = new String("Hello");


// Combined declaration and initialisation
String s2 = new String("World");


// Combined and shortened syntax form
String s3 = "Untar";
```

```
String s1 = “Yoda”;
String s2 = “Dent”;
String s3;

System.out.println(s1); // produces “Yoda”
System.out.println(s2); // produces “Dent”
```

```
String s1 = “Yoda”;
String s2 = “Dent”;
String s3;

System.out.println(s1); // produces “Yoda”
System.out.println(s2); // produces “Dent”

s3 = s2;
System.out.println(s3); // produces “Dent”
```

Compare two String

```java
// These two have the same value
new String("test").equals("test") // --> true

// ... but they are not the same object
new String("test") == "test" // --> false

// ... neither are these
new String("test") == new String("test") // --> false

// ... but these are because literals are interned by
// the compiler and thus refer to the same object
"test" == "test" // --> true

// ... string literals are concatenated by the compiler
// and the results are interned.
"test" == "te" + "st" // --> true

// ... but you should really just call Objects.equals()
Objects.equals("test", new String("test")) // --> true
Objects.equals(null, "test") // --> false
Objects.equals(null, null) // --> true
```

Other methods to compare two strings:

- `String.compareTo(String anotherString)`
- `String.compareToIgnoreCase(String anotherString)`

https://stackoverflow.com/questions/513832/how-do-i-compare-strings-in-java

Other useful methods from `String` class

- Establish how many characters there are in the text content

```
String s = “Hello, World!”;

int len = s.length();
System.out.println(len);
```

Other useful methods from `String` class

- Change the case from upper to lower and vice versa

```
String s = “Hello, World!”;

System.out.println(s.toUpperCase());

System.out.println(s.toLowerCase());
```

Other useful methods from `String` class

- Extract a sub-section, or substring of the text content

```
String s = “Hello, World!”;
String subs = s.substring(2, 5); // “llo”
```

Question #1

https://bit.ly/3B2zJrL

What is the return value of `check("madam i'm adam")`?

A. true

B. 0

C. 14

D. false

E. Tidak ada jawaban yg benar

```
public boolean check(String s) {
    String t = "";
    int len = s.length();

    for (int i = 0; i < len; i++) {
        t = s.charAt(i) + t;
    }

    return s.equals(t);
}
```

Application Programming Interface (API)

- Natural extension of the concept of a method
- Create methods with a characteristic signature of a set of formal parameters and a return value
- Invoke the method and rely on the fact that it does the job it was created to do
- We do not have to concern ourselves with how the method works unless we choose to

We have "abstracted away" the internal implementation details

Application Programming Interface (API)

- The value of an API is determined by how well it is documented so that we can understand how to use it
- The API should describe what behaviour of the library should be under all situations
- The API should only make public those features that are intended for use by the calling code

General Software Engineering Principles

Principle of Cohesion

Any single class should be designed to represent one entity, and do that job well

Principle of Encapsulation

Classes should only make public those attributes and methods that are intended for use by calling code. All other attributes and methods should remain private.

Group classes together in thematic packages

The Concept of Design Patterns

Classes should share common interfaces and interface patterns

Classes should be robust against incorrect data being fed to them and should respond in an appropriate manner by reporting errors

Documentation

Classes should be adequately documented such that they can be used by other programmers without any knowledge of their internal implementation

And it should be kept up to date with incremental improvements in the class codebase.

Javadoc

- Automatically produces class level API documentation for our code
- We need to “markup” or annotate our codebase in a particular manner
  - Commenting
  - Annotations

```java
/**
 * A class that performs some simple math operations.
 *
 * @author Kingsley Sage
 * @version 1.0
 */

public class SimpleMaths {
   /**
    * Constructor: to be developed further
    */
   public SimpleMaths() { }

   /**
    * Returns the sum of two integers.
    * @param x: an integer
    * @param y: an integer
    * @return the integer sum of x and y
    */
   public int addTwoNumbers(int x, int y) { return x + y; }
}
```

Generating Javadoc

- Via cli

```
$ javadoc filename
$ javadoc package_name
```

- IntelliJ
  - https://www.jetbrains.com/help/idea/working-with-code-documentation.html#generate-javadoc

```
-zsh
Last login: Sat Jul 17 12:01:05 on ttys006
jansonh@Jansons-MacBook-Pro ~ % vi SimpleMaths.java
jansonh@Jansons-MacBook-Pro ~ % javadoc SimpleMaths.java
Loading source file SimpleMaths.java...
Constructing Javadoc information...
Building index for all the packages and classes...
Standard Doclet version 16.0.1+0
Building tree for all the packages and classes...
Generating ./SimpleMaths.html...
Generating ./package-summary.html...
Generating ./package-tree.html...
Generating ./overview-tree.html...
Building index for all classes...
Generating ./allclasses-index.html...
Generating ./allpackages-index.html...
Generating ./index-all.html...
Generating ./index.html...
Generating ./help-doc.html...
jansonh@Jansons-MacBook-Pro ~ %
```

file:///Users/jansonh/SimpleMaths.html

PACKAGE CLASS TREE INDEX HELP

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD SEARCH: Search

Class SimpleMaths

java.lang.Object
SimpleMaths

```
public class SimpleMaths
extends Object
```

A class that performs some simple math operations.

Constructor Summary

Constructors

Constructor	Description
`SimpleMaths()`	Constructor: to be developed further

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method	Description
`int`	`addTwoNumbers(int x, int y)`	Returns the sum of two integers.

Methods inherited from class java.lang.Object

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

Constructor Details

SimpleMaths

```
public SimpleMaths()
```

Constructor: to be developed further

Method Details

addTwoNumbers

```
public int addTwoNumbers(int x,
                         int y)
```

Returns the sum of two integers.

Parameters:

x - : an integer

y - : an integer

Returns:

the integer sum of x and y

Question #2

https://bit.ly/3B2zJrL

What is the return value of `check("madam i'm adam")`?

A. -1

B. false

C. true

D. +1

E. Tidak ada jawaban yg benar

```
public boolean check(String s) {
   String t1 = "", t2 = "";
   int len = s.length(), i = 0, j = len-1;

   while (i < len && j >= 0) {
      if (Character.isLetter(s.charAt(i))
         t1 = s.charAt(i) + t1;

      if (Character.isLetter(s.charAt(j))
         t2 = s.charAt(j) + t2;

      i++; j--;
   }

   return t1.equals(t2);
}
```

The `ArrayList` Class

ArrayList

- Arbitrary size “array”
- Consist of elements of the same type
- Cannot be used directly to create collections of primitive types

```
import java.util.*;

ArrayList<String> words = new ArrayList<String>();
```

```
import java.util.*;

ArrayList<String> words = new ArrayList<String>();

words.add("crocodile");
words.add("antelope");
words.add("gnu");
words.add("zebra");
words.add("giraffe");
```

```
import java.util.*;

ArrayList<String> words = new ArrayList<String>();

words.add(“crocodile”);
words.add(“antelope”);
words.add(“gnu”);
words.add(“zebra”);
words.add(“giraffe”);

System.out.println(words.get(0)); // ”crocodile”
```

Some Key Methods

Method	Description
`boolean add(E e)`	Add an element `e` of type `E` to the end of the collection
`void clear()`	Remove all elements from the collection
`E get(int index)`	Retrieve elemnt of type `E` from position `index`
`boolean isEmpty()`	Returns `true` if the collection is empty
`E remove(int index)`	Removes the element from position `index` (returns that element of type `E`)
`int size()`	Returns the number of elements in the collection
`boolean contains(Object o)`	Returns true if the list contains the specified element `o`

```
import java.util.*;

ArrayList<String> words = new ArrayList<String>();

words.add("crocodile");
words.add("antelope");
words.add("gnu");
words.add("zebra");
words.add("giraffe");

for (String w : words) {
   System.out.println(w);
}
```

```java
import java.util.*;

// Error!
ArrayList<int> arr = new ArrayList<int>();


// Use wrapper class
ArrayList<Integer> arr = new ArrayList<Integer>();

arr.add(new Integer(15));
```

The Wrapper Classes

Primitive Type	Wrapper Class
`byte`	`Byte`
`int`	`Integer`
`long`	`Long`
`short`	`Short`
`double`	`Double`

```
Integer x1, x2;

x1 = new Integer(10);
x2 = new Integer("123");
```

```
Integer x1, x2;

x1 = new Integer(10);
x2 = new Integer("123");


int y1 = x1.intValue();
```

Question #3

https://bit.ly/3B2zJrL

Apa isi dari 5 bilangan pertama dari ArrayList `arr`?

A. 1, 2, 3, 4, 5

B. 1, 2, 4, 8, 16

C. 0, 1, 2, 4, 8

D. 2, 2, 2, 2, 2

E. Error

```
public ArrayList<Integer> generate(int maxNum) {
    ArrayList<Integer> arr = new ArrayList<>();

    int i = 1;
    while (i <= maxNum) {
        arr.add(i);
        i *= 2;
    }

    return arr;
}
```