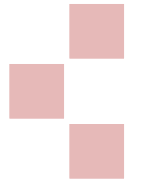
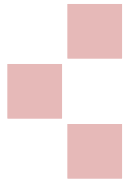


JAVA PRE-DEFINED CLASSES

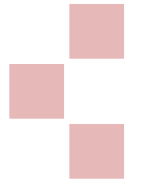
Nurochman



Java.lang package

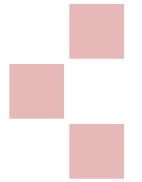
- Java compiler automatically imports all the classes in the java.lang package into every source file.
- some of the most important classes of the java.lang package:

– Object
– Math
– The wrapper classes
– String
– StringBuffer



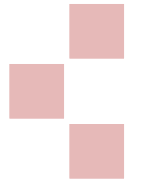
Java Pre-Defined Classes

- String
- StringBuffer
- Math
- Vector
- Hashtable
- Thread
- Socket, ServerSocket



Wrapper Classes

- Boolean
- Character
- Byte, Short, Integer, Long
- Float, Double

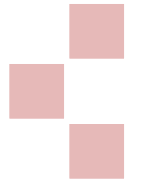


The String Class

- Class String berisi string yang tetap (immutable string).
- Sekali intance String dibuat maka isinya tidak bisa diubah.
- Memiliki beberapa konstruktor.
- Common string constructors:

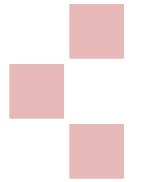
  String s1 = new String(“immutable”);

  String s1 = “immutable”;
- Java mempunyai media penyimpanan literal string yang yang disebut “pool”.
- Jika suatu literal string sudah ada di pool, Java “tidak akan membuat copy lagi”.



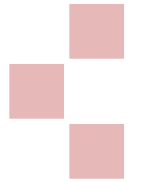
Membandingkan String

- Method equals() membandingkan contentnya
- == membandingkan alamatnya.



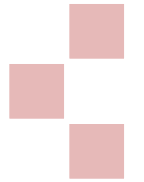
Class String Methods

- char **charAt**(int index)
- String **concat**(String str)
- static String **copyValueOf**(char[] data)
- boolean **endsWith**(String suffix)
- boolean **equals**(Object anObject)
- boolean **equalsIgnoreCase**(String anotherString)



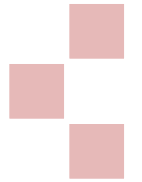
Class String Methods

- byte[] **getBytes**()
- int **indexOf**(int ch), int **indexOf**(int ch, int fromIndex)
- int **lastIndexOf**(int ch), int **lastIndexOf**(int ch, int fromIndex)
- int **length**()
- String **replace**(char oldChar, char newChar)
- String[] **split**(String regex)



Class String Methods

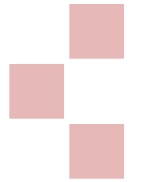
- boolean **startsWith**(String prefix)
- String **substring**(int beginIndex)
- String **substring**(int beginIndex, int endIndex)
- String **toLowerCase**()
- String **toUpperCase**()
- String **trim**()
- static String **valueOf**(boolean b), dll



The StringBuffer Class

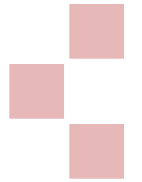
- represents a string that can be dynamically modified.
- Constructors:
  - StringBuffer(): Constructs an empty string buffer
  - StringBuffer(int capacity): Constructs an empty string buffer with the specified initial capacity
  - StringBuffer(String initialString):

    Constructs a string buffer that initially contains the specified string



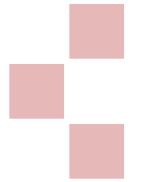
StringBuffer methods

- StringBuffer **append**(String str): Appends str to the current string buffer. Alternative forms support appending primitives and character arrays; these are converted to strings before appending.
- StringBuffer **append**(Object obj): Calls toString() on obj and appends the result to the current string buffer.
- StringBuffer **insert**(int offset, String str): Inserts str into the current string buffer at position offset. There are numerous alternative forms.



StringBuffer methods

- StringBuffer **reverse**(): Reverses the characters of the current string buffer.
- StringBuffer **setCharAt**(int offset, char newChar): Replaces the character at position offset with newChar.
- StringBuffer **setLength**(int newLength): Sets the length of the string buffer to newLength. If newLength is less than the current length, the string is truncated. If newLength is greater than the current length, the string is padded with null characters.



String Concatenation the Easy Way

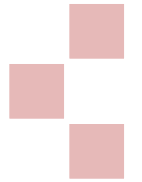
- concat() method of the String class
- append() method of the StringBuffer class
- + operator.
- Example:

  String Concatenation:

  a + b + c

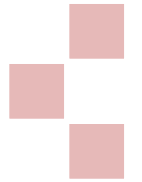
  Java compiler treats as:

  new StringBuffer().append(a).append(b).append(c).toString();



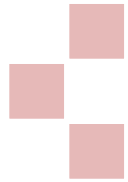
The Math Class

- a collection of methods and two constants that support mathematical computation.
- Two constans: E dan PI
- is **final**, so you cannot **extend** it.
- constructor is **private**, so you cannot create an instance.
- the methods and constants are **static**

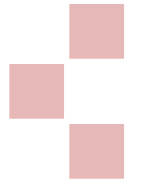


Methods of the Math Class

Method	Returns
`int abs(int i)`	Absolute value of `i`
`long abs(long l)`	Absolute value of `l`
`float abs(float f)`	Absolute value of `f`
`double abs(double d)`	Absolute value of `d`
`double ceil(double d)`	The smallest integer that is not less than `d` (returns as a `double`)
`double floor(double d)`	The largest integer that is not greater than `d` (returns as a `double`)
`int max(int i1, int i2)`	Greater of `i1` and `i2`

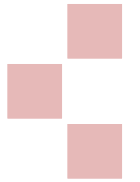


Method	Returns
`long max(long l1, long l2)`	Greater of `l1` and `l2`
`float max(float f1, float f2)`	Greater of `f1` and `f2`
`double max(double d1, double d2)`	Greater of `d1` and `d2`
`int min(int i1, int i2)`	Smaller of `i1` and `i2`
`long min(long l1, long l2)`	Smaller of `l1` and `l2`
`float min(float f1, float f2)`	Smaller of `f1` and `f2`
`double min(double d1, double d2)`	Smaller of `d1` and `d2`
`double random()`	Random number >= 0.0 and < 1.0
`int round(float f)`	Closest `int` to `f`
`long round(double d)`	Closest `long` to `d`
`double sin(double d)`	Sine of `d`
`double cos(double d)`	Cosine of `d`
`double tan(double d)`	Tangent of `d`
`double sqrt(double d)`	Square root of `d`



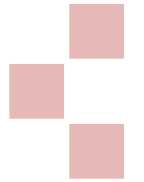
The Wrapper Classes

- is simply a class that encapsulates a
- single, immutable value.
  - Integer class wraps up an int value,
  - Float class wraps up a float value.



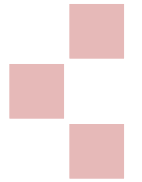
Primitives and Wrappers

Primitive Data Type	Wrapper Class
boolean	Boolean
byte	Byte
char	Character
short	Short
int	Integer
long	Long
float	Float
double	Double



Vector

- General purpose resizable array class
- Use instead of array if
  - you want to store a variable number of objects
  - all elements are objects (not basic types)
  - you need to look for an object within an array (search for an element)
  - you want to store objects of differing types

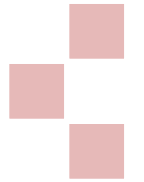


Vector VS Array

- Arrays are more efficient than vectors
- Use arrays if:
  - you have a known storage requirement

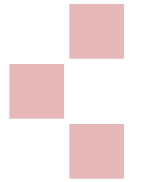
  or

  - you are only dealing with values which have the same basic type



Creating the Vector

- Vector is a class provided with the JDK
- Behaves in the same way as any other class
- Vector myVector = new Vector();
- Other constructors available



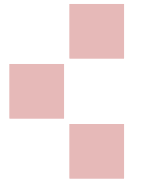
Adding Elements

void addElement(Object obj)

- Adds obj to the end of the vector

void insertElementAt(Object obj, int index)

- Adds obj at position index in the vector



Setting and Getting Elements

Object elementAt(int index)

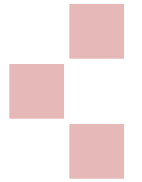
- Retrieve the object at position index

void setElementAt(Object obj, int index)

- Set obj at position index

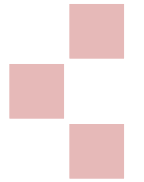
int size()

- Return size of the vector



Vector Example

```
import java.util.*;
class VectorTest {
  public static void main(String args[]){
      Vector hector = new Vector();
      hector.addElement("apple");
      hector.addElement("banana");
      hector.addElement("date");
      for(int i=0;i<hector.size();i++)
      System.out.println(hector.elementAt(i));
  }
}
```



Thread Class

void run()

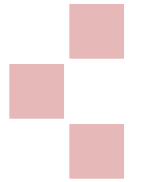
– must be overridden. Not called directly

void start()

– called to begin excution of a thread

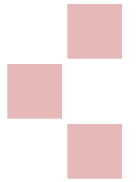
static void sleep(long m)

– makes the currently running thread pause for the given number (m) of milliseconds

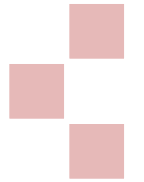


Thread Example

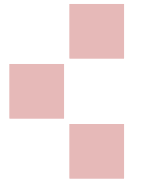
```
class ThreadExample {
    private int count=0;
    class AddThread extends Thread {
            public void run() {
                    for(int i=0;i<10;i++) {
                            count++;
                            System.out.print("Add ");
                            System.out.println(count);
                    }
            }
    }
```



```
class SubtractThread extends Thread {
        public void run() {
                for(int i=0;i<10;i++) {
                        count--;
                        System.out.print("Subtract ");
                        System.out.println(count);
                }
        }
}
```

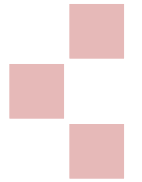


```
    public static void main(String args[]) {
        ThreadExample ex = new ThreadExample();
        AddThread a = ex.new AddThread();
        SubtractThread b = ex.new  SubtractThread();
        a.start();
        b.start();
    }
}
```



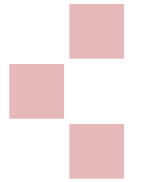
Socket

- TCP connections are made using sockets
- Connecting to port 13 of any Unix machine will give you the time
- Web servers use port 80
- Java provides Socket class for network connections



Socket Class

- Socket (String host, int port)
- synchronized void close()
- InputStream getInputStream()
- OutputStream getOutputStream()
- void setSoTimeout(int timeout)



Server Socket

- ServerSocket(int port) throws IOException
- Socket accept() throws IOException
- void close() throws IOException

Pertanyaan???