

Föreläsning 6

- Java APIs
- Objekt och referensvariabel
- Använda klasser ur klassbiblioteket

Klassbibliotek i java

Med java (JDK/JRE) följer ett stort antal **standardklasser** som man kan använda i sina program. Dessa klasser är utvecklade av programmerare på Sun.

Dessutom har andra programmerare skrivit ett stort antal klasser.

När du skriver program består stora delar av programmet av klasser som andra skrivit. Du *återanvänder* andra programmerares kod.

Många av standardklasserna ingår i något **API** (Application Programming Interface). T.ex. så samlas en hel del klasser som har med ljud, bild och video i Java Media API och klasser som hanterar användning av databaser i Java Database API.

Java API

Standardklasserna är organiserade i ***paket (package)***. Innan du använder en klass måste du i källkodsfilen ange att du ska använda klassen. Du måste importera klassen med ett import-direktiv.

Exempel: Klassen ***JOptionPane*** finns i paketet ***javax.swing***

```
import javax.swing.JOptionPane;
```

eller

```
import javax.swing.*; // Alla klasser i paketet javax.swing kan användas
```

```
public class Exemple {  
    public static void main(String[] args) {  
        JOptionPane.showMessageDialog( null, "Hello world" );  
    }  
}
```

Paketet ***java.lang*** importeras automatiskt. I paketet finns bl.a. klasserna Math, Object, Runtime, String, StringBuffer, System, Thread.

Package

Package	Provides support to
<code>java.applet</code>	Create programs (applets) that are easily transported across the Web.
<code>java.awt</code>	Draw graphics and create graphical user interfaces; AWT stands for Abstract Windowing Toolkit.
<code>java.beans</code>	Define software components that can be easily combined into applications.
<code>java.io</code>	Perform a wide variety of input and output functions.
<code>java.lang</code>	General support; it is automatically imported into all Java programs.
<code>java.math</code>	Perform calculations with arbitrarily high precision.
<code>java.net</code>	Communicate across a network.
<code>java.rmi</code>	Create programs that can be distributed across multiple computers; RMI stands for Remote Method Invocation.
<code>java.security</code>	Enforce security restrictions.
<code>java.sql</code>	Interact with databases; SQL stands for Structured Query Language.
<code>java.text</code>	Format text for output.
<code>java.util</code>	General utilities.
<code>javax.swing</code>	Create graphical user interfaces with components that extend the AWT capabilities.
<code>javax.xml.parsers</code>	Process XML documents; XML stands for eXtensible Markup Language.

Att anropa metoder i andra klasser

Du kommer att använda standardklasser på i huvudsak två sätt:

- Skapa objekt av klassen och sedan anropa en eller flera metoder
`String str = new String("Hej världen");` // skapa objekt
`int nbrOfChars = str.length();` // anropa metod

<code>int</code>	<code>length()</code> Returns the length of this string.
------------------	---

När du skapat ett objekt av klassen så anropar du en metod med hjälp av **referensvariabeln**, **punktoperatoren** och **metodens namn**.

- Använda någon form av tjänst som erbjuds av en klass. Det innebär att du anropar en metod i en klassen utan att skapa ett objekt
`JOptionPane.showMessageDialog(null, str);`

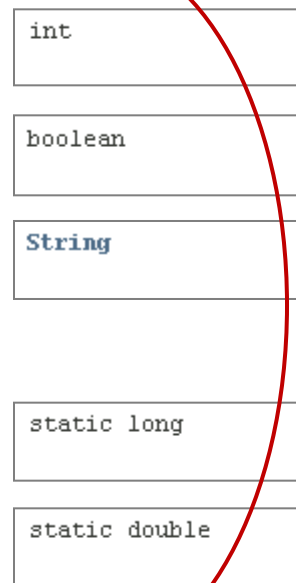
<code>static void</code>	<code>showMessageDialog(Component parentComponent, Object message)</code> Brings up an information-message dialog titled "Message".
--------------------------	--

Står det **static** i deklarationen så anropas metoden med **klassens namn**, **punktoperatoren** och **metodens namn**:
`Klassnamn.metodnamn`

Metoder och returvärde

En metod kan skrivas så den returnerar ett värde. Resultatet av anropet blir värdet metoden returnerar.

```
String str1 = "Hello world";  
String str2 = "Hej världen";  
int nbrOfChars = str1.length();           // nbrOfChars = 11;  
boolean sameContent = str1.equals( str2 ); // sameContent = false;
```



int	length() Returns the length of this string.
boolean	equals(Object anObject) Compares this string to the specified object.
String	substring(int beginIndex) Returns a new string that is a substring of this string.
static long	round(double a) Returns the closest long to the argument, with ties rounding up.
static double	sqrt(double a) Returns the correctly rounded positive square root of a double value.

Metoder och parametrar

Till vissa metoder måste man ange ett eller flera data (värden) vid anropet. Metoden behöver datan för att fungera.

```
int value1 = -10, value2;  
double realValue = 3.892;  
long roundedValue;
```

```
value2 = Math.abs( value1 );           // Math.abs( -10 );  
roundedValue = Math.round( realValue ); // Math.round( 3.892 );  
System.out.println( value2 + "," + realValue + "," + roundedValue );  
realValue = Math.sqrt( 2 );           // realValue = 1.41421...;  
roundedValue = Math.round( realValue ); // Math.round( 1.41421... );
```

static int	abs (int a) Returns the absolute value of an int value.
------------	---

static long	round (double a) Returns the closest long to the argument, with ties rounding up.
-------------	---

static double	sqrt (double a) Returns the correctly rounded positive square root of a double value.
---------------	---

String (java.lang)

Klassen **String** är till för att hantera *sekvenser av tecken*. Ett String-objekt kallas för en **sträng** i ett program.

- Ett String-objekt kan lagra noll till många tecken
- Tecknen som lagras i ett String-objekt kan inte ändras. Tecken kan inte läggas till eller tas bort. En sträng är konstant (immutable).
- Det är vanligt att det skapas ett nytt String-objekt när man anropa en metod i klassen String.

Ett String-objekt skapas med new-operatorn:

```
String str = new String("Hello world");
```

Eftersom strängar används väldigt mycket i program har man förenklat skapandet av strängar. Man kan skriva:

```
String str = "Hello world";
```

Klassen **String** är i paketet *java.lang*. Därför behöver du inte göra någon import.

String (java.lang)

När du skapat ett String-objekt kan du anropa metoder i klassen String.

```
String str1 = "Hej";
```

```
String str2 = "Goodbye";
```

```
String str3;
```

```
// Ej tilldelats något String-objekt
```

```
int len, pos;
```

```
char chr4;
```

```
// metoden length anropas för strängen "Hej". Antal tecken i "Hej" är resultatet av  
// anropet
```

```
len = str1.length();
```

```
// len = 3;
```

```
// metoden charAt anropas för strängen "Goodbye". Resultat: Tecknet i position 4
```

```
chr4 = str2.charAt(4);
```

```
// chr4 = 'b';
```

```
// Metoden substring anropas för strängen "Goodbye". Resultatet av anropet är  
// en ny sträng som innehåller alla tecken från position 0 till tecknet före  
// position 4, dvs strängen "Good"
```

```
str3 = str2.substring(0,4);
```

```
// str3 = "Good";
```

```
// metoden indexOf anropas för strängen "Good". Finns argumentet i strängen  
// blir resultatet tecknets position. Annars blir resultatet -1.
```

```
pos = str3.indexOf('d');
```

```
// pos = 3;
```

```
pos = str3.indexOf('H');
```

```
// pos = -1;
```

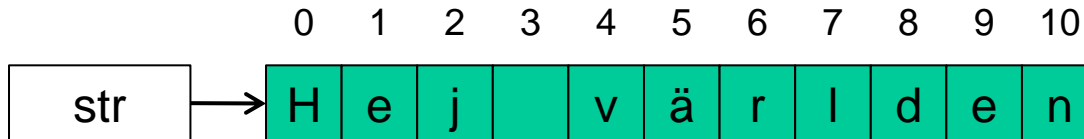
```
pos = str3.indexOf('o');
```

```
// pos = 1; första förekomsten
```

StringEx.java

String (java.lang)

```
String str = "Hej världen";
```



- *str* är en *referensvariabel* som refererar till ett String-objekt med tecknen "Hej världen".
str är en *identifierare* som vi använder när vi vill komma åt String-objektet i ett program. *str* använder vi också när vi talar om String-objektet.
- Strängen *str* har 11 tecken
- Första tecknet i *str* är i position 0
- Sista tecknet i *str* är i position 10
- Även blanktecknet är ett tecken
- Strängen *str* innehåller teckent 'j'. Det är i position 2.
- Strängen *str* innehåller inte tecknet 'V'

JOptionPane (javax.swing)

Klassen **JOptionPane** är till för du ska kunna använda dialogfönster i ett program. Dialoger kan du använda då du vill att användaren ska ge input i programmet eller då du vill visa ett meddelande för användaren.

När du använder klassen kommer du endast anropa klassmetoder.

//Inmatning av String-objekt

```
String str = JOptionPane.showInputDialog( "Mata in en text" );
```

// Inmatning av heltal

```
String str = JOptionPane.showInputDialog( "Mata in ett heltal" );  
int number = Integer.parseInt( str );
```

// Inmatning av decimaltal

```
String str = JOptionPane.showInputDialog( "Mata in ett tal" );  
double real = Double.parseDouble( str );
```

// Visa ett meddelande, 2 exempel

```
JOptionPane.showMessageDialog( null, "Hej världen" );  
JOptionPane.showMessageDialog( null, "Inmatat tal: " + str );
```

Klassen **JOptionPane** är i paketet *javax.swing*:
`import javax.swing.JOptionPane;`

Calendar (java.util)

Klassen **Calendar** är till för att *hantera datum och tid*. Ett objekt av typen Calendar får man genom ett anrop till klassmetoden *getInstance*:

```
Calendare cal = Calendar.getInstance();
```

Efter anropet refererar *cal* till ett Calendar-objekt. Calendar-objektet innehåller datorns inställningar vad det gäller datum och tid (tidpunkten då *getInstance* anropades).

Nu kan du få information från Calendar-objektet genom anrop till *get*-metoden. Som argument måste du ange vilken information du vill ha:

// Dag i månaden (1-31)

```
int day = cal.get( Calendar.DAY_OF_MONTH );           // day = 15;
```

// Klockslag – timmen i 24-timmarsformat (0-23)

```
int hour = cal.get( Calendar.HOUR_OF_DAY);           // hour = 14;
```

Ett urval av argument till *get*-metoden:

Calendar.YEAR, Calendar.MONTH, Calendar.DAY_OF_MONTH,
Calendar.Hour_OF_DAY, Calendar.MINUTE, Calendar.SECOND

```
Klassen Calendar är i paketet java.util:  
import java.util.Calendar;
```

CalendarEx.java

Math (java.lang)

Klassen **Math** är till för göra beräkningar. Klassen innehåller endast klassmetoder. Med hjälp av klassen kan man t.ex. räkna med exponenter och kvadratrötter (och mycket mer).

```
double value = 9, value2, value3, byteValue, squareroot, minimum;  
// Beräkna value*value, dvs value2  
value2 = Math.pow( value, 2 );           // value2 = 81.0;  
// Beräkna value*value*value, dvs value3  
value3 = Math.pow( value, 3 );           // value3 = 729.0;  
// Beräkna 28  
byteValue = Math.power( 2, 8 );           // byteValue = 256.0;  
// Beräkna kvadratroten av byteValue, dvs  $\sqrt{256.0}$   
squareroot = Math.sqrt( byteValue );      // squareroot = 16.0;  
// Beräkna minsta värdet av två värden  
minimum = Math.min( value2, value3 );     // jmf Math.max( v1, v2 );
```

Klassen **Math** är i paketet *java.lang*. Därför behöver du inte göra någon import.

MathEx.java

Random (java.util)

Klassen **Random** är till för kunna använda slumpvärden i ett program. Det är vanligt med slumpvärde i program då man inte vill att allt vara helt förutsägbart. Det kan handla om ett val i ett spel (datorspelare) eller vilken bild som ska visas vid en bildvisning. Klassen Random är alltså en modell av en slumpalsgenerator.

```
int rnd;
```

```
double dRnd;
```

```
// Skapa ett Random-objekt
```

```
Random randGen = new Random();
```

```
// Generera heltalssumptal
```

```
rnd = randGen.nextInt(10);
```

```
// rnd: 0 – 9
```

```
rnd = randGen.nextInt( 46 );
```

```
// rnd: 0 – 45
```

```
rnd = randGen.nextInt( 10 ) + 5;
```

```
// rnd: 5 – 14
```

```
rnd = randGen.nextInt(5) – 2;
```

```
// rnd: -2 – 2
```

```
// Generera decimala slumpstal
```

```
dRnd = randGen.nextDouble( );
```

```
// dRnd: 0 – 0.99999...
```

Klassen **Random** är i paketet *java.util*:

```
import java.util.Random;
```

RandomEx.java

Använda klassen Stopwatch

Vi har tillgång till klassen **Stopwatch**, vilken kan användas för att mäta tid. Klassen Stopwatch är alltså en modell av ett stoppur. Till klassen finns det dokumentation vilken beskriver hur klassen ska användas. Dokumentationen är genererad av programmet **javadoc.exe**.

Class Stopwatch

java.lang.Object
└─**f6.Stopwatch**

public class **Stopwatch**
extends java.lang.Object

Klassen kan användas för att mäta tid. Metoden `start()` anropas för att starta tidtagningen och metoden `stop()` anropas för att avsluta tidtagningen.

Constructor Summary

[Stopwatch\(\)](#)

Method Summary

long	getMilliSeconds() Metoden returnerar antalet millisekunder det gått mellan senaste anropet till <code>start()</code> och senaste anropet till <code>stop()</code> .
double	getSeconds() Metoden returnerar antalet sekunder det gått mellan senaste anropet till <code>start()</code> och senaste anropet till <code>stop()</code> .
void	start() Anropa <code>start()</code> då tidtagningen ska börja.
void	stop() Anropa <code>stop()</code> då tidtagningen ska avslutas.

Stopwatch.html

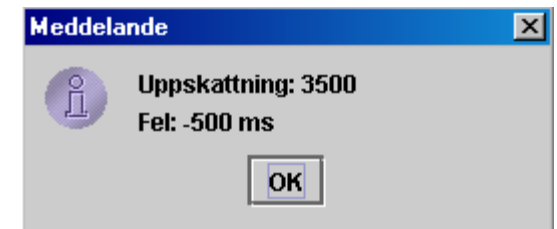
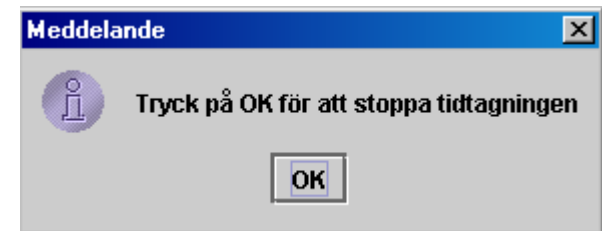
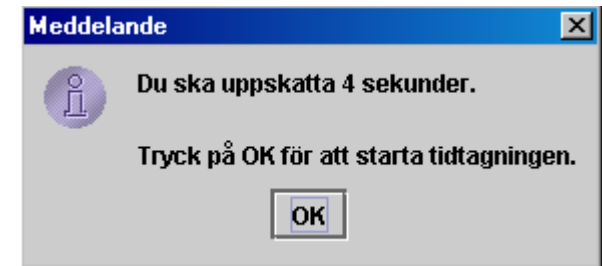
Använda klassen Stopwatch (f6)

Vi har tillgång till klassen **Stopwatch**, vilken kan användas för att mäta tid. Vi ska skriva ett program, EstimateTime, vilket låter användaren uppskatta ett visst antal sekunder.

Stopwatch
+ start():void + stop():void + getSeconds():double + getMilliseconds():long

Hur kan vi utnyttja klassen **Stopwatch**?

```
public void action() {  
    Stopwatch watch = new Stopwatch();  
    Random rand = new Random();  
    long rndTime, estimation;  
    // slumpa tid att uppskatta, 2 – 5 sek  
    // visa start-dialog  
    // starta tidtagare  
    // visa stoppa-dialog  
    // stoppa tidtagare  
    // visa resultat-dialog  
}
```



EstimateTime.java

Wrapper classes

Till samtliga primitiva datatyper finns det motsvarande klasser.

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

Det inträffar ibland att man behöver dessa objekt. Klasserna innehåller några klassmetoder och ett par konstanter som vi använder på kursen:

Klassmetoder Konstanter

Integer.parseInt Integer.MIN_VALUE, Integer.MAX_VALUE

Long.parseLong Long.MIN_VALUE, LONG.MAX_VALUE

Double.parseDouble Double.MIN_VALUE,

Double.MAX_VALUE

Wrapper-klasserna är i paketet *java.lang* och behöver därför ej importeras.