Eco-design Digitale di Base per i servizi ICT

Programmazione in Java

Massimo Giaccone, Luglio 2025

Obiettivi della lezione

- Imparare a gestire correttamente date, orari e formattazioni localizzate, con supporto per diverse lingue e culture.

Introduzione alla gestione del tempo in Java

- Java ha introdotto un nuovo pacchetto per la gestione del tempo: java.time (Java 8+)
- Sostituisce le classi Date e Calendar della vecchia API

La nuova API è più semplice, robusta e thread-safe.

LocalDate, LocalTime, LocalDateTime

```java
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;

public class Main1 {
    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        LocalDate data = LocalDate.now();
        LocalTime ora = LocalTime.now();
        LocalDateTime dataOra = LocalDateTime.now();
        System.out.println(data);
        System.out.println(ora);
        System.out.println(dataOra);
    }

}
```

Creazione di date specifiche

of() è un metodo statico che ci permette di creare oggetti data in modo leggibile.

```
LocalDate natale = LocalDate.of(year:2025, month:12, dayOfMonth:25);
```

Operazioni con le date

Le date sono immutabili: ogni operazione restituisce una nuova istanza.
Possiamo sommare o sottrarre giorni, mesi, anni...

```java
public class Main1 {
    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        LocalDate oggi = LocalDate.now();
        LocalTime ora = LocalTime.now();
        LocalDateTime dataOra = LocalDateTime.now();
        System.out.println(data);
        System.out.println(ora);
        System.out.println(dataOra);

        LocalDate natale = LocalDate.of(year:2025, month:12, dayOfMonth:25);
        LocalDate domani = oggi.plusDays(daysToAdd:1);
        LocalDate meseProssimo = oggi.plusMonths(monthsToAdd:1);
    }

}
```

Periodo e ChronoUnit

Period rappresenta una quantità di tempo in giorni, mesi, anni.

ChronoUnit permette di calcolare direttamente il numero di unità temporali tra due date.

```java
import java.time.LocalDate;
import java.time.Period;
import java.time.temporal.ChronoUnit;

public class Main1 {
    Run | Debug | Run main | Debug main
    public static void main(String[] args) {

        LocalDate dataInizio = LocalDate.now();
        LocalDate dataFine = LocalDate.EPOCH;
        Period periodo = Period.between(dataInizio, dataFine);
        long giorni = ChronoUnit.DAYS.between(dataInizio, dataFine);
    }

}
```

Parsing e formatting con DateTimeFormatter

DateTimeFormatter è molto flessibile e supporta formati personalizzati.
Possiamo anche usarlo per fare il parsing da stringhe a date.

```
public class Main1 {
    Run | Debug | Run main | Debug main
    public static void main(String[] args) {

        LocalDate data = LocalDate.now();
        DateTimeFormatter fmt = DateTimeFormatter.ofPattern(pattern:"dd/MM/yyyy");
        String formato = data.format(fmt);
    }

}
```

Esercizio 1

Scrivere un programma che calcoli i giorni tra due date inserite come LocalDate.

Introduzione all’internazionalizzazione

- Internazionalizzazione = adattare il software a diverse lingue/aree

- Java supporta locale, formati numerici, data e messaggi

Locale

Locale definisce lingua e nazione. È usato per scegliere i formati e i messaggi più adatti all'utente.

```java
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.Locale;

public class Main1 {
    Run | Debug | Run main | Debug main
    public static void main(String[] args) {

        Locale it = new Locale(language:"it", country:"IT");
        Locale us = new Locale(language:"en", country:"US");
    }

}
```

DateFormat e formattazione localizzata

DateFormat restituisce una rappresentazione della data diversa in base al locale scelto. Utile per mostrare la data in formato comprensibile all'utente.

```java
import java.text.DateFormat;
import java.util.Locale;

public class Main1 {
    Run | Debug | Run main | Debug main
    public static void main(String[] args) {

        Locale it = new Locale(language:"it", country:"IT");
        Locale us = new Locale(language:"en", country:"US");

        DateFormat df = DateFormat.getDateInstance(DateFormat.LONG, Locale.ITALY);
    }

}
```

Esercizio 2

Stampare la data corrente in tre formati:

- Italiano
- Inglese USA
- Giapponese

NumberFormat

NumberFormat gestisce numeri, valute e percentuali in modo localizzato. Utile per mostrare importi secondo la cultura dell'utente.

```
public class Main1 {
    Run | Debug | Run main | Debug main
    public static void main(String[] args) {

        Locale it = new Locale(language:"it", country:"IT");
        Locale us = new Locale(language:"en", country:"US");

        DateFormat df = DateFormat.getDateInstance(DateFormat.LONG, Locale.ITALY);
        NumberFormat nf = NumberFormat.getCurrencyInstance(Locale.FRANCE);
    }

}
```

Esercizio 3

- Stampare una cifra in formato valuta per diversi paesi (IT, US, DE). Aggiungere la data localizzata.

Riepilogo gestione date

- Usa java.time per date e orari
- LocalDate e DateTimeFormatter sono la base
- Immutabilità = sicurezza

Riepilogo internazionalizzazione

- Locale, NumberFormat, DateFormat, ResourceBundle
- Separazione tra codice e testo
- Adattabilità a nuovi mercati