Programmazione II

A.A. 2022-23
Prof. Maria Tortorella

- La classe File
- Ancora sulle classi con input e con output
- Ancora sulle classi PrintStream e Scanner

La classe Name con output

- Sono già state trattate le classi con output
- Ogni classe deve avere almeno un metodo print

```
// Nella classe Name

void print() {
      System.out.println("titolo: " + this.title);
      System.out.println("name: " +this.name);
      System.out.println(this.surname);
}
```

Poco flessibile: stampa solo su video

La classe Name con output

- Si potrebbe passare lo stream come argomento

```
void print(PrintStream target) {
      target.println(this.title);
      target.println(this.name);
      target.println(this.surname);
}
```

È possibile memorizzare lo stato degli oggetti su file

Ha bisogno di un oggetto della classe PrintStream

La classe File

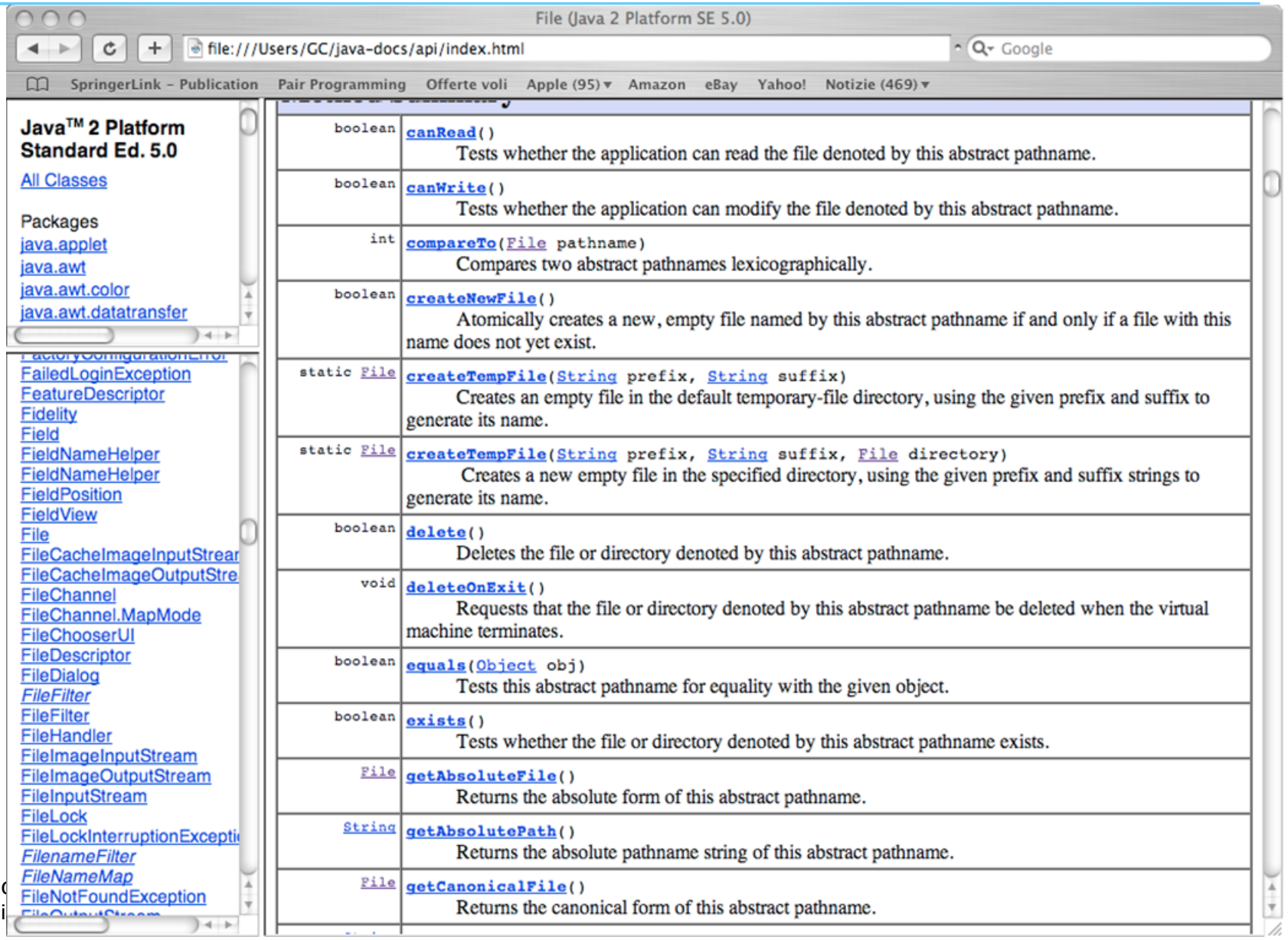
Bisogna utilizzare la classe File
per modellare path-name di file e directory

Costruttore

File(String `pathname`)

- Creates a new `File` instance by converting the given pathname string into an abstract pathname.

La classe File



La Classe PrintStream

Constructor Summary
PrintStream(File file) Creates a new print stream, without automatic line flushing, with the specified file.
PrintStream(File file, String csn) Creates a new print stream, without automatic line flushing, with the specified file and charset.
PrintStream(OutputStream out) Create a new print stream.
PrintStream(OutputStream out, boolean autoFlush) Create a new print stream.
PrintStream(OutputStream out, boolean autoFlush, String encoding) Create a new print stream.
PrintStream(String fileName) Creates a new print stream, without automatic line flushing, with the specified file name.
PrintStream(String fileName, String csn) Creates a new print stream, without automatic line flushing, with the specified file name and charset.

Uso del metodo print

```
class Name{
….
    void print(PrintStream target) {
        target.println(this.title);
        target.println(this.name);
        target.println(this.surname);
    }
….
}
```

```
class TesterName{
….
    static void main(String[] arr){
        ….
        PrintStream ps =
                new PrintStream(new File("prova.txt");
        Name n = new Name("Mario","Rossi");
        n.print(ps);
        n.print();
        …
    }
….
}
```

Classi con input

Analogamente la classe deve essere con input

- Obiettivo: inviare un messaggio ad una classe chiedendo la creazione di un nuovo oggetto di quella classe
- Ad esempio, nella classe Name: chiedere alla classe Name di creare un oggetto della stessa classe

```
class TesterName{
….
    static void main(String[] arr){
        ….
        PrintStream ps = new PrintStream(new File("prova.txt");
        Scanner sc = new Scanner(System.in);
        Name n = Name.read();
        n.print(ps);
        …
    }
….
}
```

Leggere un Name

- I dati possono essere su file:

```
Mario Rossi
Filippo Verdi
Paolo Bianchi
```

```
public static Name read(Scanner s) {
        String name, surname;
        name = s.next();
        surmane = s.nextLine();
        return new Name(name, surname);
}
```

```
...
Scanner sc = new Scanner(new File("fileinput.txt"));
Name nome = Name.read(sc);

....
nome.print(System.out);
```

La classe Scanner

Constructor Summary
Scanner(File source) Constructs a new Scanner that produces values scanned from the specified file.
Scanner(File source, String charsetName) Constructs a new Scanner that produces values scanned from the specified file.
Scanner(InputStream source) Constructs a new Scanner that produces values scanned from the specified input stream.
Scanner(InputStream source, String charsetName) Constructs a new Scanner that produces values scanned from the specified input stream.
Scanner(Readable source) Constructs a new Scanner that produces values scanned from the specified source.
Scanner(ReadableByteChannel source) Constructs a new Scanner that produces values scanned from the specified channel.
Scanner(ReadableByteChannel source, String charsetName) Constructs a new Scanner that produces values scanned from the specified channel.
Scanner(String source) Constructs a new Scanner that produces values scanned from the specified string.

Verifica di fine input

- Meglio controllare la disponibilità del dato in input prima di leggerlo
  - Un metodo di lettura può restituire null per indicare che non ci sono dati in un file
- Il metodo read per la classe Name

```
public static Name read(Scanner sc) {
      String first, last;
      if (!sc.hasNext ()) return null;
      first = sc.next();
      if (!sc.hasNextLine ()) return null;
      last = sc.nextLine();
      return new Name(first,last);
}
```

Il metodo **hasNext** restituisce true o false in base alla disponibilità del dato
Esistono anche **hasNextLine, hasNextInt, hasNextBoolean, ...**

Verifica di fine input

- Se l'input è da tastiera si deve verificare se il dato letto è corretto
  - Il metodo di lettura può restituire null se il dato non è valido
- Leggere da tastiera un oggetto della classe Name

```
public static Name read() {
      Scanner sc=new Scanner (System.in);
      String first, last;
      first = s.next();
      if (first.equals(""))
            return null;
      last = s.next();
      if (first.equals(""))
            return null;
      return new Name(first,last);
}
```

La classe Truck

- Aggiungere un metodo di lettura ed uno scenario di utilizzo

Trucks.txt

```
2 15900
6 20990
5 17000
```

```
public static Truck read(Scanner sc) {
      if (!sc.hasNextInt()) return null;
      int axles = sc.nextInt();
      if (!sc.hasNextInt()) return null;
      int weight = sc.nextInt();
      return new Truck(axles,weight);
}
```

- Modificare la classe TollBooth per gestire il fatto che caselli diversi possano usare valori diversi per il costo per asse e per tonnellata

Esempio:
Retribuzione dei dipendenti

- Torniamo al problema della retribuzione dei dipendenti
- Presentazione del problema

  Modellare un sistema di retribuzione per dipendenti che sono pagati con una tariffa oraria.

  Il sistema deve riuscire a calcolare la retribuzione di un dipendente sulla base della tariffa oraria e delle ore di lavoro effettuate e deve stampare il nome, le ore e la paga calcolata. I dipendenti che lavorano più di 40 ore ricevono una somma per gli straordinari, pagati una volta e mezzo la loro tariffa salariale normale. Se un dipendente ha 30 o più ore di straordinario nelle ultime due settimane viene emesso un messaggio d'avviso

La classe Employee

```
public class Employee{
  // methods
public Employee(String name, int rate) {
      this.name = name;
      this.rate = rate;
      this.lastWeeksOvertime = 0;
  }
  public int calcPay(int hours) {
      ...
  }
  public String getName() {
       return this.name;
  }
  // instance variables
  private String name;
  private int rate;
  private int lastWeeksOvertime;
}
```

Il calcolo degli stipendi

- È possibile dotare la classe Employee del metodo di input – di lettura

```
Mario Rossi
20
Paolo Verdi
25
Anna Bianchi
24
...
```

```
public static Employee read(Scanner s) {
        String name;
        if (!s.hasNextLine())
                return null;
        name = s.nextLine();
        int rate;
        if (!s.hasNextInt())
                return null;
        rate = s.nextInt();
        s.nextLine();
        return new Employee(name, rate);
}
```

Il calcolo degli stipendi

- E se ci sono più impiegati per i quali calcolare lo stipendio?
- Si può utilizzare una struttura di controllo ciclica while (già nota)
  - Ecco il ciclo per il calcolo degli stipendi degli impiegati

```
Employee e = Employee.read(scf);
while (e != null) {
      int hours = s.nextInt();
      System.out.println("Employee " + e.getName() +
            " has earned " + e.calcPay(hours));
      e = Employee.read(scf);
}
```

- La condizione (e != null) termina il ciclo quando non ci sono più dati nel file di ingresso.

Versione scorretta

- La seguente versione è scorretta

```
Employee e;
while (e != null) {
      e = Employee.read(scf);
      int hours = sc.nextInt(br.readLine());
      System.out.println("Employee " + e.getName() +
                  " has earned " + e.calcPay(hours));
}
```

- Notare che nella condizione si verifica il valore della variabile senza che questa sia stata inizializzata
- La compilazione causa un errrore *"uninitialized variable*"

Un'altra versione scorretta

```
Employee e = Employee.read(scf);
while (e != null) {
      int hours = sc.nextInt(br.readLine());
      System.out.println("Employee " + e.getName() +
                  " has earned " + e.calcPay(hours));
}
```

- In questo caso vengono letti solo i dati del primo impiegato – non c'è nessun Employee.read nel corpo del ciclo
- Il ciclo quindi non fa progressi verso la terminazione (*ciclo infinito* -- una volta entrati non si esce più !!!)

Loop patterns

- Altri task di lettura e calcolo ripetitivi sono simili nella forma al ciclo per il calcolo degli stipendi
- Per esempio, per calcolare i pedaggi dei Truck che arrivano ad un casello:

```
TollBooth tollBooth = new TollBooth("Benevento");
Truck truck;
truck = Truck.read(scf);      // read first
while (truck != null) {
      tollBooth.calculateToll(truck);     // processing
      truck = Truck.read(scf)             // read next
}
```

Loop patterns

- Tutti i cicli precedenti hanno la stessa forma

```
// Loop Pattern: read/process
read first item
while (a valid item has been read) { // as long as an item
                                     // was successfully obtained
        process the item
        read the next item
  }
```

- Un tale frammento di codice è chiamato *loop pattern*, in quanto può essere usato per creare strutture di ciclo
- Questo in particolare è chiamato *read/process loop pattern*

Read/Process Loop Pattern

- Per usare questo pattern, bisogna definire:
  - Il metodo per leggere l'oggetto
  - La condizione che indica che un oggetto valido è stato letto
  - Il codice per elaborare l'oggetto

Libreria di canzoni

- Problema
  - WOLD, una stazione radio locale, vuole informatizzare la propria libreria di canzoni.
  - Si è creato uno o più file in cui sono stati inseriti degli elementi composti dai titoli e dai compositori delle canzoni.
  - Si intende dare al disc-jockey la possibilità di cercare nella libreria tutte le canzoni di un particolare artista.

Libreria di canzoni

- Scenario d'esempio

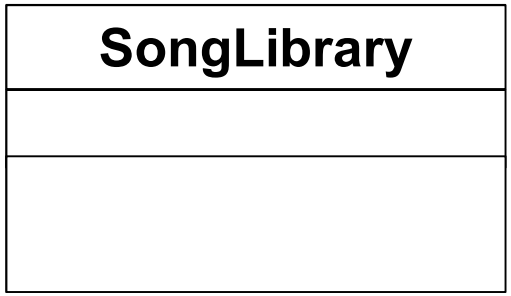
Inserisci il nome del file della libreria di canzoni: **ClassicRock.lib**
File ClassicRock.lib loaded.
Inserisci l'artista da cercare: **Beatles**
Canzoni dei Beatles trovate:
Back in the USSR
Paperback writer
She Love You
Inserisci l'artista da cercare: **Mozart**
Nessuna canzone di Mozart trovata

Libreria di canzoni

- Problema
  - WOLD, una stazione radio locale, vuole informatizzare la propria libreria di canzoni.
  - Si è creato uno o più file in cui sono stati inseriti degli elementi composti dai titoli e dai compositori delle canzoni.
  - Si intende dare al disc-jockey la possibilità di cercare nella libreria tutte le canzoni di un particolare artista.

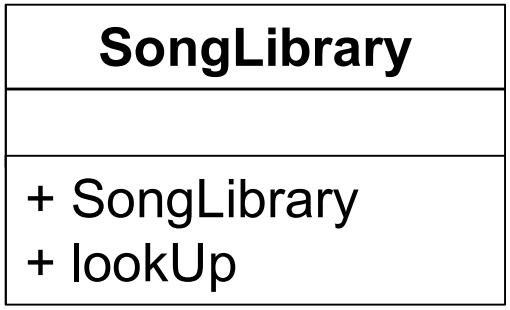
Oggetti primari

- Sostantivi: song library, song, file, entry, title, artist
- Artist e title sono parti di song, che è sussidiaria di song library
- File e entry (in un file) rappresentano solo dati da leggere
- Classi: SongLibrary e Song



Comportamento desiderato

- Capacità di creare una SongLibrary
  - Costruttore
- Necessità di cercare le canzoni di un artista
  - Un metodo lookUp



Un esempio:
Libreria di canzoni

- Interfaccia – La classe SongLibrary
- La creazione di una SongLibrary richiede l'indicazione del file che memorizza i brani musicali di un certo genere
  - SongLibrary("classical.lib")
  - La ricerca dei brani di un certo autore richiede il nome dell'autore
  - lookUp("Gould");

SongLibrary
+ SongLibrary(songFileName: String) + lookUp(artist: String): void

Variabili di istanza e costruttore

- Ogni volta che viene invocato lookUp crea un nuovo Scanner associato al file su disco specificato dal nome del file di canzoni (passato al costruttore).
- Questo nome deve quindi essere mantenuto in una variabile d'istanza

Variabile d'istanza

SongLibrary
- songFileName: String
+ SongLibrary(songFileName: String) + lookUp(artist: String): void

Traduzione

SongLibrary
- fileName: String
+ SongLibrary(fileName: String) + lookUp(artist: String): void

```
class SongLibrary {
      public SongLibrary(String songFileName) {
            ...
      }
      public void lookUp(String artist) throws Exception
        {
            ...
        }
      private String songFileName;
   }
```

Definire l'interfaccia

- Tipico codice di utilizzo

```
SongLibrary classical = new SongLibrary("classical.lib");
SongLibrary jazz = new SongLibrary("jazz.lib");
classical.lookUp("Gould");
classical.lookUp("Marsalas");
jazz.lookUp("Corea");
jazz.lookUp("Marsalas");
```

Variabili di istanza e costruttore

- La variabile d'istanza è inizializzata dal costruttore

```
class SongLibrary {
   public SongLibrary(String songFileName) { // costruttore
            this.songFileName = songFileName;
   }
      ...   // Metodo lookup
   private String songFileName;     // variabile d'istanza
}
```

Metodo lookup

```
public void lookUp(String artist) throws Exception {

    File fileIn = new File(songFileName);
    Scanner fileInScanner = new Scanner(fileIn);

    Song song = Song.read(fileInScanner);
    while(song != null) {
            if (artist.equals(song.getArtist()))
                System.out.println(song.getTitle());
            song = Song.read(fileInScanner);
      }
  }
```

La classe Song

- L'interfaccia e le variabili d'istanza

Song
- title: String - artist: String
+ Song(title: String, artist: String) + read(fileName: String):Song + getTitle(): String + getArtist(): String + print():

IMPLEMENTARE PER ESERCIZIO

Esercizio: La classe Time

- *Costruire una classe Time che permetta di manipolare valori temporali. Ci restringiamo a periodi di 24 ore, richiedendo così solo ore e minuti*
- Una classe Time è un esempio di classe *utility* o *helper* – una che è usata da altre classi
- Comportamenti richiesti alla classe Time:
  - *Confrontare due orari*: dati due orari, indicare quale orario precede e quale segue
  - *Aggiungere un intervallo di tempo*: dato un orario di partenza e un intervallo di tempo, calcolare l'orario in cui termina l'intervallo
  - *Stampare un orario*

La classe Time

- Nel nostro scenario è stato individuato un insieme utile di operazioni sui valori temporali che costituiranno il comportamento della classe:
  - Time (constructor)
  - addDuration
  - isBefore
  - isAfter
  - print

La classe Time

```
Time t1 = new Time(10, 15, "am");
Time t2 = new Time(3, 10, "pm");
Time t3 = t2.addDuration(30);   //add 30 minutes
t3.print(System.out);           //prints 3:40pm
System.out.println();
Time t4 = t2.addDuration(3, 30); // add 3 hours, 30 min
t4.print(System.out);              // prints 6:40pm
System.out.println();
```

La classe Time

```
Time earlier;
Time later;
if (t3.isBefore(t4)) {
          earlier=t3;
          later=t4;
} else {
          earlier=t4;
          later=t3;
}
earlier.print(System.out);
System.out.print(" is earlier than ");
later.print(System.out);
System.out.println();
```

La classe Time

```
class Time {
  // Methods
  public Time(int hours, int minutes, String
  amOrPm) {...}
  public Time addDuration(int minutes) {...}
  public Time addDuration(int hours, int minutes)
      {...}
  public boolean isBefore(Time t){...}
  public boolean isAfter(Time t){...}
  public void print(PrintStream target){...}
  // Instance Variables
  ...
}
```

Overloading del metodo addDuration

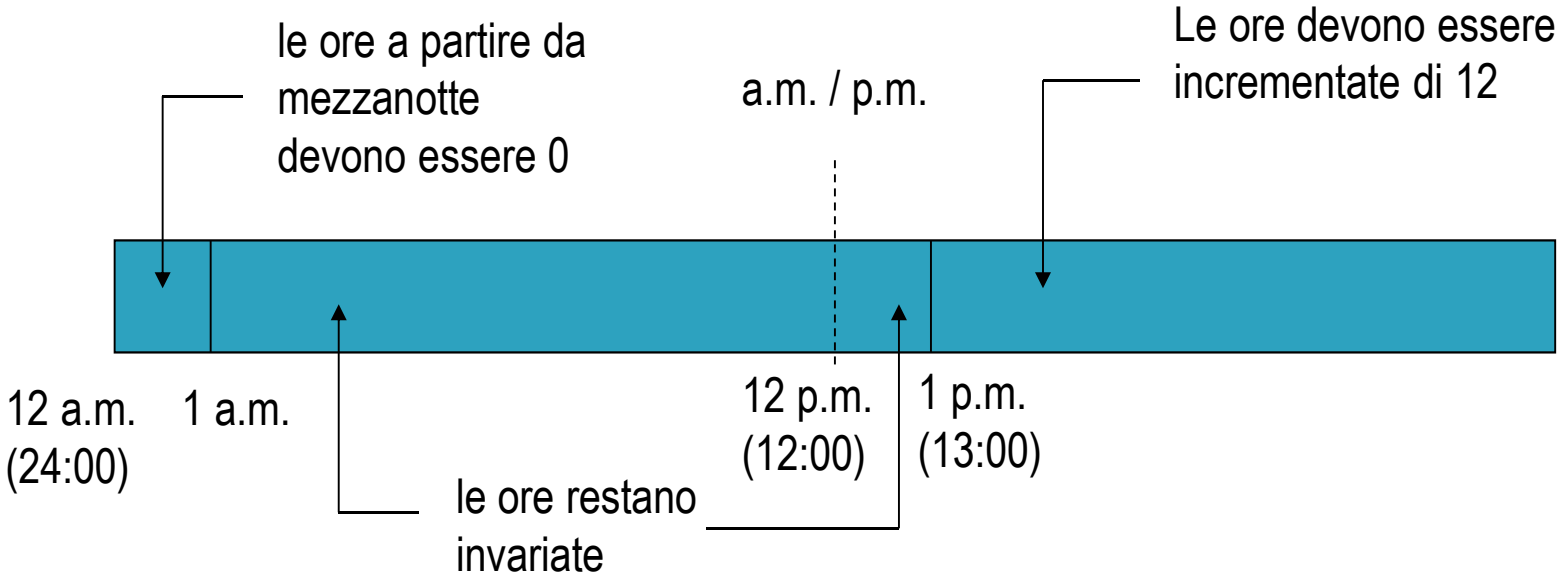
La classe Time

- Rappresentiamo il tempo con il numero totale di minuti

```
class Time {
    // Methods
    ...
    // Instance variables
    int totalMinutes;
}
```

La classe Time

- Il costruttore deve convertire la coppia ore/minuti nel numero totale di minuti
- A tal fine bisogna dividere il giorno in sezioni



La classe Time

- Completare l'implementazione della classe Time come esercizio

Esercizio

- Scrivere una classe capace di monitorare l'andamento di un investimento nel tempo assumendo un tasso di interesse annuale fisso

Esercizio

- Un file "Studenti.dati" contiene un elenco di studenti, organizzato come segue
  Nome Cognome Matricola Media AnnoIscrizione FuoriCorso
- Esempio

Paolo	Bianchi	1511627	28 .3	2	true
Mario	Rossi	1234563	24.7	1	false
Maria	Verdi	12427459	20.0	3	true

- Scrivere un programma che
  - identifichi lo studente con la media massima
  - Identifichi lo studente con la media minima, supposti unici,
  - Listi gli studenti che sono fuori corso
  - Listi gli studenti iscritti all'anno 2

Esercizio

- Il file "Studenti.dati" contiene un elenco di studenti, organizzato come segue
  - Nome Cognome Matricola
- Il file "Esami.dati" contiene un elenco di esami superati, organizzato come segue
  - NomeEsame Voto Matricola
- Scrivere un programma che, letto da tastiera nome e cognome di uno studente (supposto unico) ne calcoli e visualizzi la media

Esercizio

- Class Point
  java.awt.Point

- Dati due punti, P1 e P2, ed un valore reale X, trovare quanto vale la retta che passa per i punti P1 e P2 nel punto di ascissa X.

Esercizio

- Un file, “funzione.dati” contiene una funzione campionata, un punto per riga, in ordine crescente delle ascisse.
- 13.63 16.54
- 16.56 12.74
- …
- Scrivere un programma che calcoli il valore della funzione in un punto dato X
  - Interpolazione lineare nei due punti + vicini …