PROGRAMMAZIONE A OGGETTI
Pattern, buone pratiche e cose da ricordare.

1) Esempio di override del metodo equals, da fare per ogni classe

```
public boolean equals(Object oggetto){

      ClasseMia altro=(ClasseMia)oggetto;

      if(altro.attributo==this.attributo && altro.attributo1==this.attributo1)
            return true;
      else
            return false;
}
```

N.b. ovviamente l'if deve contenere ogni attributo della mia classe

2) Il metodo contains restituisce un booleano, come anche il metodo equals. Invece il metodo **compareTo** compara le stringhe e restituisce un **intero:**

- **maggiore di zero se la stringa viene dopo il parametro in ordine alfabetico**
- **minore di zero se la stringa viene prima il parametro in ordine alfabetico**
- **uguale a zero se le stringhe sono uguali.**

3) Metodi di stampa per LinkedList (accesso al singolo elemento della lista col metodo get(indice))

- FOR

```
for(int i=0; i<lista.size(); i++){
      ClasseMia ctemp= lista.get(i);
      System.out.println(ctemp.toString());
}
ma anche più brevemente
for(int i=0; i<lista.size(); i++){
      System.out.println(lista.get(i).toString());
}
```

- FOR-EACH meglio se non ho bisogno di far qualcosa in particolare con il contatore

```
for(ClasseMia ctemp: lista){
      System.out.println(ctemp.toString());
}
```

- per le classi che implementano collection, implementano anche ITERABLE quindi possiamo scrivere anche la stampa sfruttando l'ITERATORE
  - con il while

```
Iterator<ClasseMia> it= lista.iterator(); //se uso il tipo generico non devo
fare downcast
            int j=1;
            while(it.hasNext()) {
                  System.out.println("Numero "+ j);
                  System.out.println(it.next().toString());
                  j++;
            }
```

- con il do while(il cast è necessario)

```
Iterator it= lista.iterator();
            do {
                  ClasseMia ctemp=(ctemp)it.next();
                  System.out.println(ctemp);
            }
            while(it.hasNext());
```

4) le mappe spesso nell'avere una chiave richiedono che un dato venga duplicato, tuttavia se i dati memorizzati come valori sono parecchi l'accesso per chiavi è conveniente e veloce, per aggiungere elementi alla mappa **non** uso add ma uso **put.** Quando devo usare una mappa come per le collezioni non distanzio map ma i suoi "figli" che sono classi e non interfacce treemap ed hashmap, inoltre è importante ricordare che **nelle mappe non ci vanno i tipi generici ma solo oggetti (es no int ma Integer).**

5) Per stampare il contenuto di una mappa HashMap senza le parentesi ma elemento per elemento devo prima trasferire i valori in una linked list e poi stamparli per esempio con un foreach

```
//data una Hashmap chiamata m
LinkedList l= new LinkedList(m.values());
for(ClasseMia ctemp: l){
      System.out.println(ctemp.toString());
}
```

6) se uso una treemap ho l'ordine alfabetico delle chiavi, se ho bisogno dell'ordine di inserimento conviene utilizzare una lista

7)Quando vanno implementati dei comandi di associazione e ci sono delle liste all'interno delle classi devi sempre implementare in ciascun programma il comando di aggiunta delle istanze alla lista con una apposita funzione aggiungiElementoDaAggiungere e se c'è da effettuare una sostituzione da fare di volta in volta va implementata una funzione di rimozione da inserire nel setElementoDaSostituire vincolata (la rimozione) da un if x!=null

8) quando si implementa l'interfaccia COMPARABLE e di conseguenza il compareTo per ottenere un ordinamento in ordine alfabetico NORMALE rispetto a più parametri, bisogna restituire la comparazione PRIMA dell'elemento locale con this, paragonato all'elemento passato come parametro (es c). In questo modo l'ordine alfabetico sarà naturale.
Per la questione dei più parametri uso struttura if-else in cui nell'else c'è l'ordinazione per parametro "secondario"

```
public int compareTo(ClasseMia c) {

            int diff= this.nome.compareTo(c.nome);;
            if(diff!=0)
                  return diff;
            else
                  return this.cognome.compareTo(c.cognome);
      }
```

9) se devi aggiornare un codice in una TreeMap o lista della classe principale allora va fatto con una variabile intera DIRETTAMENTE nei metodi della classe principale, se invece il numero va tenuto in elenchi interni ad altre classi allora il numero va tenuto lì.

10) per avere una mappa che mantenga l'ordine di inserimento si possono usare le LinkedHashMap senza problemi, uguali alle altre mappe.

11) ricerca di un elemento specifico secondo certe condizioni tramite un for-each con lo scatenarsi di un'eccezione

```
ClasseMia c= null;
            for(ClasseMia ctemp : mappadellamiaclasse.values()){
//esempio di una condizione di ricerca
                  if(ctemp.getAttributo1().compareTo(parametro1)==0 &&
ctemp.getAttributo2().compareTo(parametro2)==0) {
                        c=ctemp;
                        break;

                  }

            }
            //assicurati che non sia nullo perché altrimenti non si scatena
l'eccezione!
            if(c!=null)
                  return c;
            else throw new EccezioneClasseMiaInesistente();
```

12) se devo ordinare dividendo per sottoclassi creo una lista per ciascuna sottoclasse, poi le unisco in un'unica lista del tipo della superasse nell'ordine in cui mi servono

Esempio tratto dal tema d'esame sui tornai di calcio in cui si richiede di ordinare dirigenti e calciatori in una stessa lista mettendo prima i dirigenti poi i calciatori tutti e due rispettivamente ordinati per nome e cognome (vedi sopra)

```
      public Collection<Tesserato> elencoTesseratiSquadra(String nomeSquadra){
            LinkedList<Tesserato> listaTesserati= new LinkedList<Tesserato>();
            LinkedList<Calciatore> clista= new LinkedList<Calciatore>();
            LinkedList<Dirigente> dlista= new LinkedList<Dirigente>();

            for(Tesserato t: tesserati.values()) {
                  if(t instanceof Dirigente ) {

if (((Dirigente) t).getSquadra().getNome().compareTo(nomeSquadra)==0)
                              dlista.add((Dirigente)t);
                  }
                  }

            for(Tesserato t: tesserati.values()) {
                  if(t instanceof Calciatore) {

if (((Calciatore) t).getSquadra().getNome().compareTo(nomeSquadra)==0)
                              clista.add((Calciatore)t);
                  }
                  }


            Collections.sort(clista);
            Collections.sort(dlista);
            listaTesserati.addAll(dlista);
            listaTesserati.addAll(clista);
            return listaTesserati ;
      }
```

13) se hai metodi interni alla classe principale si possono usare DIRETTAMENTE nella stessa per risparmiare righe di codice ed essere più precisi scrivendo semplicemente il nome della

```
Arbitro a= (Arbitro)cercaTesseratoPerNomeCognome(nomeArbitro, cognomeArbitro);
```

14) Più che usare i comparatori in classi a parte, usarli direttamente nella classe da ordinare in questo modo, parametrizzando il comparatore della classe e usando già oggetti del tipo giusto in compare

```
 static class ComparatorePerGiornata implements Comparator<Incontro> {

            public int compare(Incontro i1, Incontro i2) {
                  return (i1.getGiornata()-i2.getGiornata());
            }
      }
```

il richiamo all'interno del software si fa così, citando la classe in cui è presente la classe statica di comparazione

```
Collections.sort(incontriPerGiornata, new
Incontro.ComparatorePerDifferenzaReti());
```

15) Per ordinare dei numeri in ORDINE CRESCENTE (in questo caso i numeri erano double, per cui il cast a int) si fa una semplice differenza:

```
public int compareTo(Prenotazione p) {
            return (int)(this.importo-p.importo);
      }
```

16) Per ordinare dei numeri in ORDINE DECRESCENTE (erano sempre double) davanti alla differenza si mette MENO

```
public int compareTo(Pratica p) {
            return (int)(-(this.importoTotale-p.importoTotale));
      }
```

17) Richiesta di ordinazione per data CRESCENTE con data nel formato YYYYMMDD, analizzando date di sottoclassi diverse (metodo con inserimento della classe che implementa il comparator nella classe da ordinare)

```
static class ComparatorePerData implements Comparator<Prenotazione>{

            public int compare(Prenotazione p1, Prenotazione p2) {

if(p1 instanceof PrenotazioneAlbergo && p2 instanceof PrenotazioneAlbergo) {
      int data1= Integer.parseInt(((PrenotazioneAlbergo)p1).getDataCheckIn());
      int data2= Integer.parseInt(((PrenotazioneAlbergo)p2).getDataCheckIn());
      return data1-data2;

                  }
else if(p1 instanceof PrenotazioneAlbergo && p2 instanceof PrenotazioneVolo) {
      int data1= Integer.parseInt(((PrenotazioneAlbergo)p1).getDataCheckIn());
      int data2= Integer.parseInt(((PrenotazioneVolo)p2).getDataPartenza());
      return data1-data2;
                  }
else if(p1 instanceof PrenotazioneVolo && p2 instanceof PrenotazioneAlbergo) {
      int data1= Integer.parseInt(((PrenotazioneVolo)p1).getDataPartenza());
      int data2= Integer.parseInt(((PrenotazioneAlbergo)p2).getDataCheckIn());
      return data1-data2;
                  }
else {
      int data1= Integer.parseInt(((PrenotazioneVolo)p1).getDataPartenza());
      int data2= Integer.parseInt(((PrenotazioneVolo)p2).getDataPartenza());
      return data1-data2;
                  }
            }

      }
```

richiamo di questo metodo:

```
public Collection<Prenotazione> elencoPrenotazioniPerData()
      {
      Collections.sort(prenotazioni, new Prenotazione.ComparatorePerData());
            return prenotazioni;
      }
```

18) gestione della lettura di un file RIGA PER RIGA, con gestione delle eccezioni

```
      public void leggiFile(String nomeFile) throws IOException {
            BufferedReader br= new BufferedReader(new FileReader(nomeFile));
            String riga;
            while((riga= br.readLine())!=null) {
                  String s[]= riga.split(";");
            try {
                  try {//eccezione rispetto ad altre richiuse nel file
                              }
                        catch(ProdottoInesistenteException e) {
                                    e.printStackTrace();
                        }
                  }

            }
            catch(NoSuchElementException e) {
                        e.printStackTrace();
                  }
            }
            br.close();
      }
```

19) aggiornamento di una variabile in una mappa che conteneva già alcuni di quei valori; in questo caso **Nel caso l'utente recensisca lo stesso hotel due volte nella stessa giornata, il sistema sovrascrive la recensione precedentemente memorizzata e restituisce un riferimento all'oggetto di tipo Recensione aggiornato (il sistema permette invece allo stesso utente di effettuare recensioni in date diverse, ad esempio in caso di soggiorni multipli presso lo stesso hotel).**

```
      public Recensione aggiungiRecensione(String data, String titolo, String
testo, double voto, String username, int codiceHotel){
            Hotel h= hotel.get(codiceHotel);
            Utente u= utenti.get(username);
            codiceRecensione++;
            Recensione r= new Recensione(codiceRecensione, data, titolo,
testo, voto, u, h);
            for(Recensione rtemp: recensioni.values()) {
                  if(rtemp.getData().compareTo(data)==0 &&
rtemp.getHotel().getCodice()==codice &&
rtemp.getUtente().getUsername().compareTo(username)==0) {
                        r= new Recensione(rtemp.getCodice(), data, titolo,
testo, voto, u, h);
                        codiceRecensione--;
                  }
            }
            recensioni.put(r.getCodice(), r);
            return r;
      }
```

20) conversione SICURA di double in int per un comparatore decrescente

```
public int compareTo(Hotel o) {
            int media1= (int)(this.calcolaMedia()*1000);
            int media2= (int)(o.calcolaMedia()*1000);
            return -(media1-media2);
      }
```

22) lettura da file con try catch e string tokenizer

```
  public void leggi(String file) {

      try
      {
            BufferedReader br = new
                 BufferedReader(new FileReader(file));
            String riga;
            while( (riga = br.readLine())!=null  )
            {
                  try
                  {
                        // Qui ho letto una riga

                        StringTokenizer st = new StringTokenizer(riga,";");
                        // Non genero un array di stringhe come con split(),
ma accedo ai singoli "token" mediante il metodo nextToken();

                        String type = st.nextToken();
                        String nome  = st.nextToken();
                              etc.
                        }
                  }
                  catch(EditoreInesistente e){
                        eccezione riferita ai metodi usati sopra
                  }
                  catch(NoSuchElementException e){
                   eccezione riferita ad un problema di uso delle stringhe
                  }
                  catch(NumberFormatException e){
                        problema di formato numerico nelle conversioni a
integer e double
                  }

            }
      }
      catch(FileNotFoundException e){

      }
      catch(IOException e){

      }

    }
```

21) gestione di vendite per settimane e mesi comparatore di emergenza!!

```
int[] settimane = new int[52];
      int[] mesi = new int[12];


public void registraVendita(int settimana, int mese){
      settimane[settimana-1]++;
      mesi[mese-1]++;
      this.quantita--;

      if(this.quantita==this.soglia)
      {
           libreria.creaOrdine(this, quantitaRiordino);
      }

    }

public Collection getClassificaMese(final int mese){
      List<Libro> classifica = new LinkedList<Libro>( libri );
      Collections.sort(classifica,new Comparator<Object>(){
            public int compare(Object x, Object y){
                Libro a=(Libro)x;
                Libro b=(Libro)y;
                return - (a.mesi[mese-1]-b.mesi[mese-1]);
            }
      });
      return classifica;
    }
```