Commenti sull'esercizio 2

La prima cosa da fare era realizzare una sottoclasse PizzaMap della classe data FixedArrayMap. La classe specializza la mappa in modo che contenga associazioni tra i nomi delle pizze e i rispettivi prezzi in euro. Era richiesto di implementare due metodi. Riporto qui sotto le specifiche e i commenti relativi a ciascuno di essi.

public Object put(Object key, Object value)
Innanzitutto la firma del metodo era data. Qualsiasi implementazione di un metodo con firma diversa, es Object(String key, Double value) è sbagliata perché non sovrascrive il metodo.

Il metodo put verifica i pre-requisiti per una mappa che memorizza associazioni tra il nome della pizza (una parola) e il prezzo associato (di tipo in grado di gestire numeri con la virgola).
Molti non hanno inserito **nessun tipo di controllo** sui pre-requisiti. Era necessario invece scrivere:

```
if ((key==null) || (value==null) ||
    !(key instanceof String) || !(value instanceof Double)){
                throw new IllegalArgumentException();
}
```

Alcuni hanno controllato solo che key fosse una stringa, in questo caso i requisiti sono stati considerati parzialmente soddisfatti. Nel caso specifico andava bene anche non verificare che i riferimenti fossero null:

```
if(!(key instanceof String) || !(value instanceof Double)){
     throw new IllegalArgumentException();
}//ok
```

Questo perché instanceof restituisce false se la variabile contiene null, ma concettualmente non fa male esplicitarlo anche perché in caso di altri pre-requisiti può essere necessario. Ad esempio se ho una stringa s e devo verificare se la sua lunghezza è 0, se non controllo prima se s è diverso da null potrei avere NullPointerException scrivendo if(s.length()==0).

Il metodo put di FixedArrayMap lancia l'eccezione FullMapException se la mappa è piena. In questo caso l'eccezione deve essere catturata e gestita ridimensionando opportunamente l'array p, prima di inserire la nuova associazione.
Uno degli obiettivi di questo esercizio era vedere quanto si è capito dell'ereditarietà. Il metodo put di FixedArrayMap fa già tutto ciò di cui ho bisogno. Questo include la rimozione di un'eventuale associazione già presente nella mappa (quindi non c'è nessun bisogno di invocare esplicitamente i metodi remove o get di FixedArrayMap) e l'inserimento della nuova associazione.

L'unico aspetto da curare è la possibilità che l'array p (gestito da FixedArrayMap a dimensione fissa) si riempia, nel qual caso put di FixedArrayMap lancerà FullMapException. **Era chiesto esplicitamente di gestirla per vedere se il concetto di lancio-cattura dell'eccezione era stato compreso**. Per questo è importante rispettare le consegne. Chi non ha gestito l'eccezione ma inserito direttamente in p la nuova associazione (copiando sostanzialmente il contenuto di put di FixedArrayMap dopo il ridimensionamento di p) non ha risposto alla consegna. Il codice per gestire l'eccezione doveva essere del tipo:

```
Object result=null;
try{
      result = super.put(key,value);
}
catch(FullMapException e){
      p = resize(p,2*pSize);
      result = super.put(key,value);
}
```

Era importante, dopo il ridimensionamento di p, invocare nuovamente super.put, infatti se la sua invocazione nel try lancia l'eccezione significa che l'inserimento non c'è stato. Era altrettanto importante assicurarsi di ridimensionare p. Qualcuno ha creato la copia di dimensione doppia inserendo correttamente nelle prime posizioni il contenuto di p, ma poi non ha assegnato il nuovo riferimento a p, lasciando questo array alla dimensione originale.

Il valore restituito è il valore precedentemente associato alla data chiave, se già presente, oppure null, se la chiave è nuova nella mappa.
Questo risultato si ottiene direttamente dall'invocazione di put di FixedArrayMap. Io ho salvato il risultato in una variabile e poi restituito il suo valore, ma andava bene anche restituire direttamente il valore di put quando lo si invocava nel try e nel catch scrivendo direttamente: `return super.put(key,value);`
Qualcuno ha restituito null a prescindere e questo è un errore perché non rispetta la consegna.

Un errore che mi è particolarmente dispiaciuto vedere (dopo essermi raccomandata tanto a lezione) è la ridefinizione delle variabili d'istanza della classe FixedArrayMap in PizzaMap:

```
public class PizzaMap extends FixedArrayMap {
      private Pair[] menu;
      private int menuSize;
      …
}
```

oppure la definizione di una variabile oggetto di tipo FixedArrayMap dentro PizzaMap:

```
public class PizzaMap extends FixedArrayMap{

      private FixedArrayMap rist;
      public PizzaMap(){
            rist=new FixedArrayMap();
      }
      …
}
```

Quando si crea una sottoclasse si ereditano tutte le variabili e i metodi della superclasse da cui la si deriva. Quello che può succedere è che non possiamo accedere direttamente a tutte le variabili e/o metodi della superclasse (dipende dal qualificatore d'accesso public/protected/private). Quando in classe abbiamo creato SavingAccount come sottoclasse di BankAccount non abbiamo ridefinito la variabile balance perché ogni oggetto di tipo SavingsAccount avrà la sua copia di balance come già definito dalla superclasse. Analogamente, dentro SavingAccount non abbiamo creato un oggetto BankAccount b = new BankAccount() per invocare i metodi della superclasse! Li invochiamo tramite il prefisso "super." prima del nome del metodo. Questi concetti sono fondamentali per la programmazione orientata agli oggetti. Raccomando a tutti (anche a chi ha superato l'esame ma sa di aver sbagliato qui) di ripassarli bene!

Sugli **algoritmi di ordinamento** non mi dilungo, li trovate descritti nelle slide delle lezioni. Molti hanno **sbagliato qualche indice** (ad esempio nell'insertion sort il primo indice avanza, mentre il secondo deve essere decrementato, mentre più di qualcuno ha incrementato entrambi). Nel mergesort qualcuno ha creato i due array left e right ma **ha scordato di popolarli con gli elementi dell'array** di partenza prima di invocare ricorsivamente l'ordinamento su di essi. Questi errori vengono di solito segnalati in esecuzione con dei NullPointerException o ArrayIndexOutOfBound exception ed è quindi possibile correggerli.
Inoltre era **richiesto esplicitamente di non modificare l'array p**. Chi l'ha fatto non ha rispettato la consegna. Bisognava creare una copia dell'array p che avesse dimensione fisica uguale alla dimensione logica di p e quindi contenesse tutti e soli gli elementi di p. Più di qualcuno **ha usato nei cicli di ordinamento la dimensione fisica di p** (andando a valutare celle che contenevano riferimenti null).

Un aspetto importante è che dovevate ordinare oggetti di tipo Pair in base al contenuto della variabile value. C'erano diversi modi di fare questo, li ho considerati buoni tutti purché funzionassero. Gli errori principali qui sono stati di confronto diretto tra gli elementi dell'array da ordinare:

```
if(newP[j] < newP[min]) oppure
if(newP[j].compareTo(newP[min])<0)
```

Il primo è concettualmente sbagliato perché confronta i riferimenti. Il secondo è corretto solo se si implementa il metodo compareTo nella classe Pair in modo che restituisse -1/0/1 in base ai valori di value dei due oggetti. Questo non era richiesto (anzi la classe FixedArrayMap non si sarebbe dovuta modificare), ma se qualcuno l'ha fatto l'ho considerato ok.

Anche il confronto `if(newP[j].getValue() < newP[min].getValue())` è sbagliato perché getValue restituisce un Object. Va fatto il casting su Double o il downcasting su double per fare un confronto tra i numeri in virgola mobile contenuti nell'oggetto:
`if((Double)(newP[j].getValue())<(Double)(newP[min].getValue()))`
Più di qualcuno ha fatto casting su String (non mi è chiaro perché, assumo un po' di confusione). In questo caso avete però ClassCastException… sbagliare è umano ma usate i messaggi di compilatore e interprete per correggere!

Il tester è stato implementato in generale abbastanza bene. Gli errori più frequenti sono stati la lettura dei due dati da inserire con next(). Questo metodo restituisce una stringa, quindi:

```
PizzaMap pizzeria = new PizzaMap();
// qua ci va il codice per aprire il file con gestione obbligatoria delle eccezioni
// …

while(scan.hasNext()){
      pizzeria.put(scan.next(), scan.next());
}
```

lancerà IllegalArgumentException perché in PizzaMap c'è (o dovrebbe esserci) un controllo per verificare che il **secondo argomento sia di tipo Double**. Quindi per non avere errori si sarebbe dovuto scrivere:

```
while(scan.hasNext()){
      pizza.put(scan.next(), scan.nextDouble());
}
```

Anche in questo caso però può verificarsi un'eccezione nel caso in cui il file di input non sia ben formattato. Infatti **hasNext() verifica che ci sia un token successivo da leggere, non che ce ne siano due**! Pertanto è possibile che la chiamata per il secondo argomento lanci **NoSuchElementException**, che pochi hanno gestito.

Per la stampa della mappa e del menù era sufficiente invocare i metodi corrispondenti ricordandosi che restituiscono una stringa e che quindi devono essere invocati come argomento di un System.out.println(). Non servivano cicli che andassero a invocare metodi di accesso.

La stampa dello scontrino richiedeva l'accesso tramite i metodi già forniti al valore associato alle pizze indicate. Il metodo get di FixedArrayMap restituisce un Object, pertanto andava fatto casting su Double o double per fare la somma.

Per quanto riguarda la lettura di nomi di pizza composti da più parole trovate diverse possibili versioni nelle soluzioni che ho fornito.

Per quanto riguarda la formattazione del prezzo totale con due cifre dopo la virgola bisognava utilizzare il metodo printf (visto con un esempio a lezione e poi anche con i TJ in laboratorio):

System.out.printf("Prezzo per una margherita, una bufala e due pizze al prosciutto: %.2f \n",tot);

Qualcuno ha stampato dopo il risultato un 0… questa è una soluzione ad hoc per il valore dell'esempio. Se la somma fosse stata 20.55 l'aggiunta dello 0 avrebbe stampato 20.550. In una delle soluzioni da voi proposte lo 0 veniva aggiunto solo dopo aver esaminato il totale (trasformato in una stringa) e verificato che effettivamente servisse. Questa soluzione è quantomeno corretta, anche se molto macchinosa (e fantasiosa!).