Programmazione ad Oggetti 2011 - 2012
Esercizi su Generics e Double Dispatch

Collections e Generics

Un predicato logico un enunciato che dipende da uno o più argomenti. Il predicato assume un valore di verità (vero o falso) ogni volta che si fissano i valori degli argomenti. Ad esempio, dato linsieme dei numeri naturali, possiamo definire il predicato

p(x) = "x è un numero dispari"

Applicando il predicato a diversi numeri naturali otteniamo diversi valori di verità: ad esempio p(7) vero, mentre p(10) è falso. In Java, possiamo realizzare il concetto di predicato mediante "oggetti predicato" descritti dalla seguente interfaccia :

```
interface Predicate
{
     public boolean test(Object obj);
}
```

L'idea è che tutte le classi che implementano questa interfaccia realizzano un particolare predicato mediante il metodo `test()`.

1. Definite una classe `Predicates` che implementi i seguenti metodi statici. I metodi utilizzano un oggetto `Predicate` per eseguire diverse operazioni su una collezione.

```
/**
 * Rimuove da coll ogni oggetto per cui il predicato pred e' vero
 */
 public static  void remove(Collection coll, Predicate  pred)

/**
 * Rimuove da coll ogni oggetto per cui il predicato pred e' falso
 */
 public static  void retain(Collection  coll, Predicate pred)

/**
 * Restituisce una lista che contiene tutti gli oggetti per cui
 * il predicato pred e' vero
 */
 public static  List collect(Collection coll, Predicate  pred)
```

```
/**
 * Restituisce l'indice del primo elemento di list per cui
 * il predicato pred e' vero. Se pred e' falso per tutti gli
 * elementi di list  restituisce -1;
 */
public static int find(List list, Predicate pred)
```

2. Modifichiamo linterfaccia Predicate utilizzando i generics, come segue

```
interface GenPredicate<T>
{
    public boolean test(T obj);
}
```

Definite una classe `GenPredicates` che includa le versioni generiche dei metodi della classe `Predicates` che avete sviluppato in precedenza (utilizzando `GenPredicate` al posto di `Predicate`, e opportune versioni generiche dei parametri di tipo collezione). Scegliete la firma dei metodi nella nuova classe in modo da assicurare un adeguato grado di flessibilità nellutilizzo dei metodi.

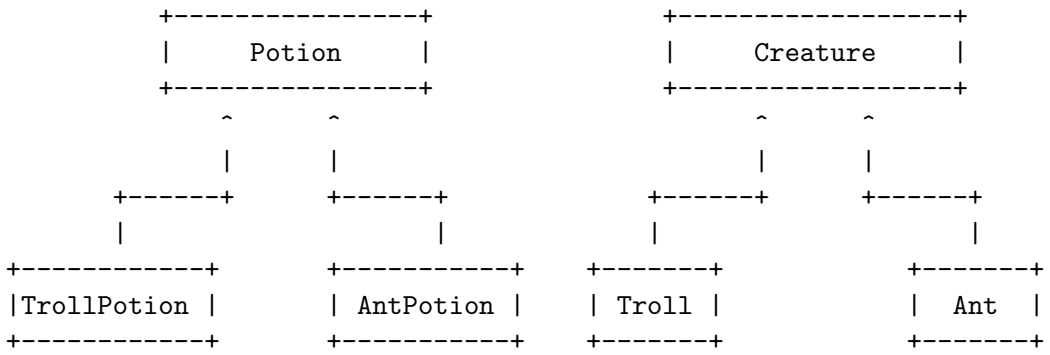
Ad esempio, la vostra definiizione dovrebbe permettere le seguenti invocazioni:

```
List<Integer>  li = new ArrayList<Integer>();
GenPredicate<Integer> pi = . . .
GenPredicate<Number> pn = . . .
GenPredicates.retain(li, pi);
GenPredicates.remove(li, pn);
List<Number> ln = GenPredicates.collect(li, pn);
int i = GenPredicates.find(ln,pn);
```

Definite una classe di test per i metodi di `GenPredicates` e testate in modo estensivo la vostra implementazione.

Double Dispatching

Sono date le seguenti due gerarchie di classi che rappresentano, in forma estremamente semplificata, le entità di un *adventure game* elettronico.



Vogliamo rappresentare il fatto che diversi tipi di pozioni hanno diversi effetti sui diversi tipi di creature: ad esempio, una `TrollPotion` potrebbe uccidere un `Troll` ma non avere effetti su una `Ant`, e corrispondentemente una `AntPotion` potrebbe uccidere una `Ant` e non avere effetto sulle altre creature. Possiamo modellare questo effetto definendo per la classe `Potion` una metodo `spray()` secondo il seguente schema:

```
class Potion
{
   public void spray( Creature aCreature ) { . . .  }

   . . .
}
```

e realizzando l'implementazione in modo che il metodo si comporti diversamente a seconda del tipo di pozione e della creatura.

Completate l'implementazione della classe `Potion` e di tutte le classi della gerarchia in modo da ottenere l'effetto desiderato, **senza utilizzare il predicato** `instanceof` e quindi **senza utilizzare espliciti testcases sul tipo degli oggetti coinvolti**.

Testate la vostra implementaizone utilizzando la seguente classe test:

```
class potions
{
    public static void main(String[] args)
    {

        Potion container = null;
        Creature aCreature = null;

        // test Potion on different creatures
        container = new Potion();

        aCreature = new Creature();
        container.spray( aCreature );

        aCreature = new Troll();
        container.spray( aCreature );

        aCreature = new Ant();
        container.spray( aCreature );

        // test TrollPotion on different creatures
        container = new TrollPotion();

        aCreature = new Creature();
        container.spray( aCreature );

        aCreature = new Troll();
        container.spray( aCreature );
```

```
        aCreature = new Ant();
        container.spray( aCreature );

        // test AntPotion on different creatures
        container = new AntPotion();

        aCreature = new Creature();
        container.spray( aCreature );

        aCreature = new Troll();
        container.spray( aCreature );

        aCreature = new Ant();
        container.spray( aCreature );
    }
}
```

e assicurandovi che l'output prodotto sia il seguente: L'esecuzione del metodo `main()` deve fornire il seguente output.

```
Potion sprayed on Creature
Potion sprayed on Troll
Potion sprayed on Creature
TrollPotion sprayed on Creature
TrollPotion sprayed on Troll
TrollPotion sprayed on Creature
AntPotion sprayed on Creature
AntPotion sprayed on Troll
AntPotion sprayed on Creature
```