Conjuntos de inteiros

$\mathbb{ZZZZZZZZ}$

Conjuntos de inteiros

- Os conjuntos de inteiros têm diversas aplicações em muitos domínios
  - Números do totoloto
  - Números de telefone
  - ...
- Pretendemos construir diversos tipos de conjuntos de inteiros, para depois escolher o mais adequado consoante as situações
  - Em particular, queremos
    - Um conjunto simples de inteiros
    - Um conjunto de inteiros em que seja fácil saber qual o maior
    - Um conjunto de inteiros ordenado

O nosso programa deve permitir

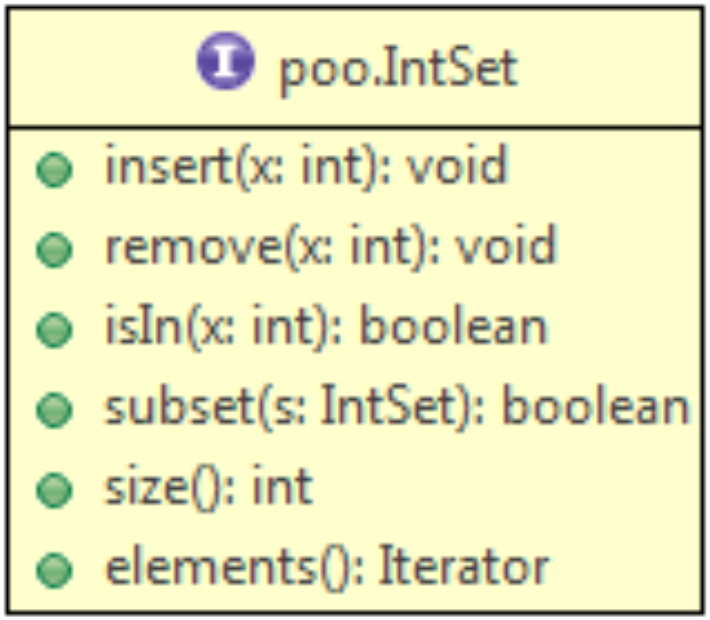
- Criar um conjunto de inteiros
  - Simples, ou com a funcionalidade extra de encontrar rapidamente o máximo
- Acrescentar números inteiros a um conjunto
- Remover um número inteiro do conjunto
- Testar se um número pertence ao conjunto
- Testar a relação de subconjunto
- Listar os elementos de um conjunto

Vamos definir uma família de conjuntos de inteiros

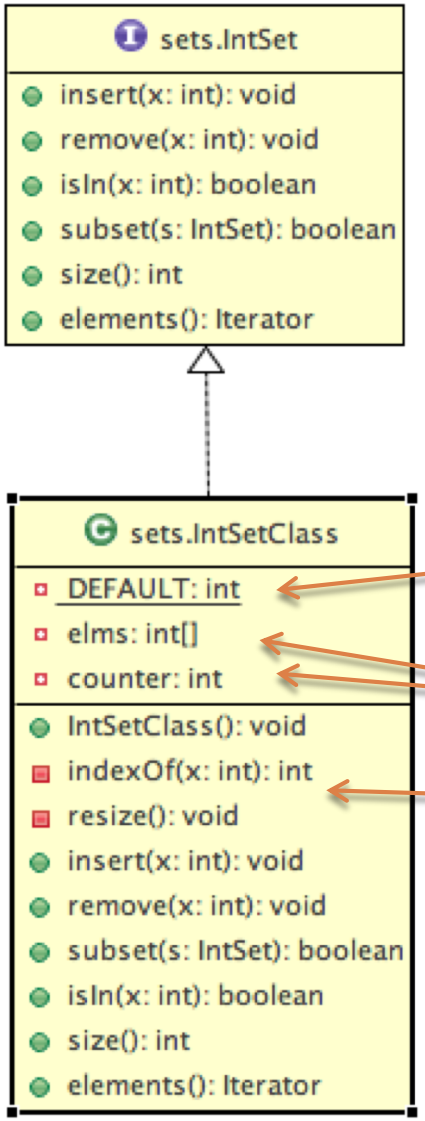
- `IntSet` fornece um conjunto adequado de métodos para conjuntos alteráveis, não limitados, de inteiros
  - **`public void`** `insert(int x)`
    - Acrescenta o inteiro `x` ao conjunto **PRE:** `!isIn(x)`
  - **`public void`** `remove(int x)`
    - Remove o inteiro `x` do conjunto **PRE:** `isIn(x)`
  - **`public boolean`** `isIn(int x)`
    - Se `x` pertence ao conjunto, retorna `true`, caso contrário, retorna `false`
  - **`public boolean`** `subset(IntSet s)`
    - Se **`this`** é sub-conjunto de `s` retorna `true`, caso contrário, retorna `false`
  - **`public int`** `size()`
    - Retorna o tamanho do conjunto
  - **`public`** `Iterator elements()`
    - Retorna um iterador de inteiros, para suportar as listagens

A interface IntSet


```
// Comentários omitidos por economia de espaço no slide :-(
public interface IntSet {
  public void insert(int x);
  public void remove(int x);
  public boolean isIn(int x);
  public boolean subset(IntSet s);
  public int size();
  public Iterator elements();
}
```

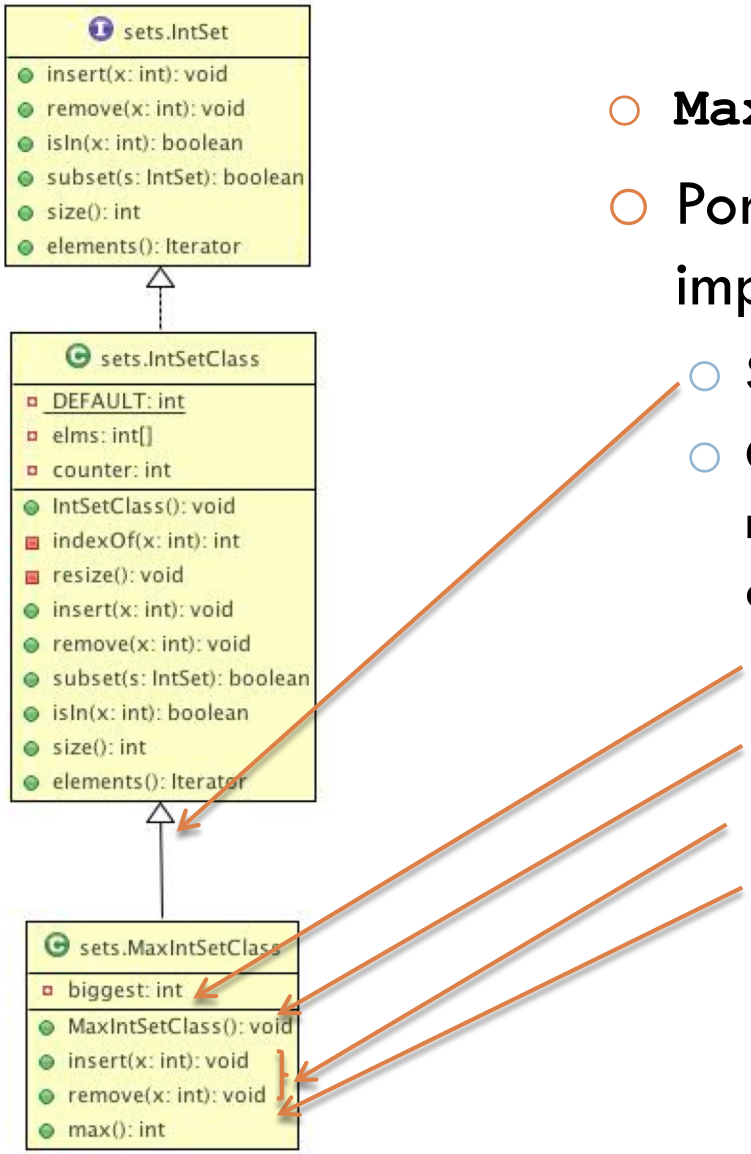


A classe IntSetClass



- Vamos definir uma classe que implemente a interface `IntSet` de modo a criar Conjuntos de inteiros simples
  - Esquema de implementação da interface a que já estamos habituados
    - Acrescentamos uma constante, com o tamanho por omissão
    - Duas variáveis, com um vector acompanhado
    - Dois métodos auxiliares, privados, já nossos conhecidos...

Uma família de conjuntos de inteiros

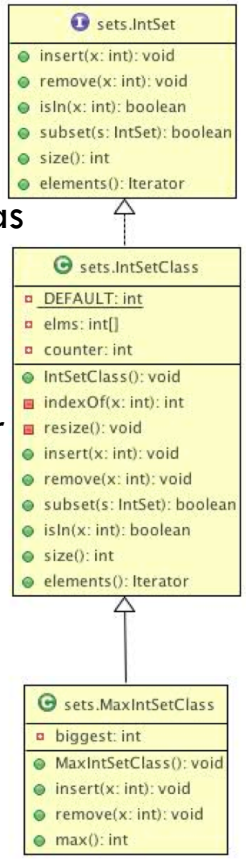


- **`MaxIntSet`** é uma sub-classe de **`IntSetClass`**
- Por ser sub-classe de **`IntSetClass`**, também implementa a interface **`IntSet`**:
  - Símbolo de herança
  - Comportamento semelhante ao de **`IntSetClass`**, mas com um método extra **`max`** que retorna o maior elemento do conjunto
    - Variável **`biggest`** acrescentada
    - Novo construtor
    - **`insert`** e **`remove`** redefinidos
    - Método **`max`** acrescentado

Especificação de `IntSetClass`


```
public class IntSetClass implements IntSet {

  public IntSetClass(){}

  public void insert(int x) {...}
  public void remove(int x) {...}
  public boolean isIn(int x) {...}
  public boolean subset(IntSet s) {...}
  public int size(){...}
  public Iterator elements() {...}
}
```

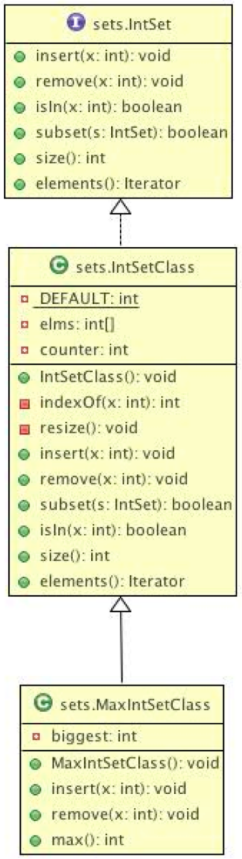
Não são usados membros protegidos, neste exemplo. Isto significa que as sub-classes de IntSetClass apenas lhe podem aceder através da sua interface pública. O nível de acesso é aceitável, porque o iterador permite visitar todos os elementos da colecção.



Implementação de `IntSetClass`

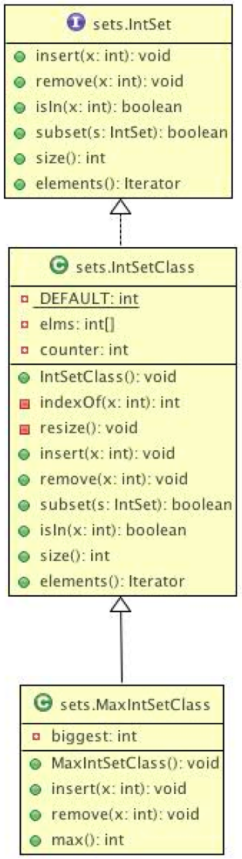

```
public class IntSetClass implements IntSet {
  private static final int DEFAULT = 10;
  private int[] elms;
  private int counter;

  public IntSetClass(){...}
  private int indexOf(int x) {...}
  private void resize() {...}
  public void insert(int x) {...}
  public void remove(int x) {...}
  public boolean isIn(int x) {...}
  public boolean subset(IntSet s) {...}
  public int size(){...}
  public Iterator elements() {...}
}
```

Esta constante, as duas variáveis e os método `indexOf` e `resize` são privados.
São inacessíveis fora desta classe, mesmo para as sub-classes.



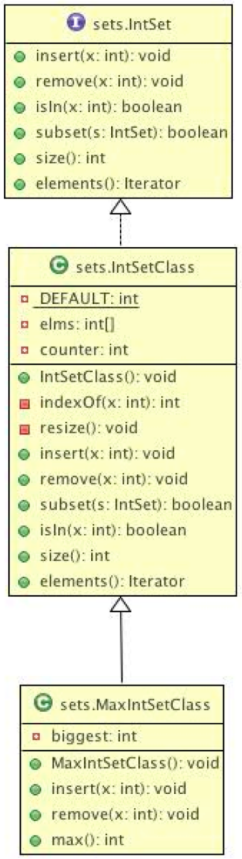
Implementação de `IntSetClass`


```
public class IntSetClass implements IntSet {
  private static final int DEFAULT = 10;
  private int[] elms;
  private int counter;

  public IntSetClass(){
    elms = new int[DEFAULT];
    counter = 0;
  }

  private int indexOf(int x) {
    int i = 0;
    while (i < counter) {
      if (elms[i]==x)
        return i;
      i++;
    }
    return -1;
  }
```



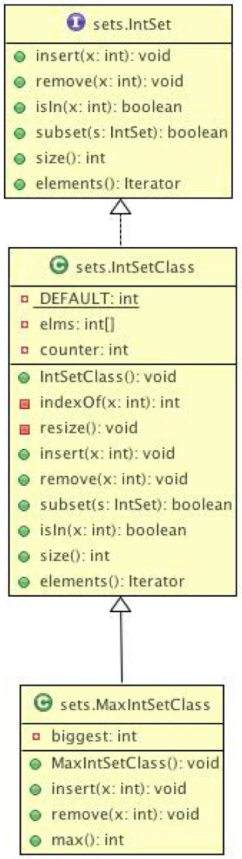
Implementação de `IntSetClass`


```
public void insert(int x) {
  if (counter == elms.length)
     resize();
  elms[counter++] = x;
}

private void resize() {
  int[] tmp = new int[elms.length*2];
  for (int i = 0; i < counter; i++)
      tmp[i] = elms[i];
  elms = tmp;
}

public void remove(int x) {
  int index = indexOf(x);
  counter--;
  elms[index] = elms[counter];
}
```



Implementação de `IntSetClass`

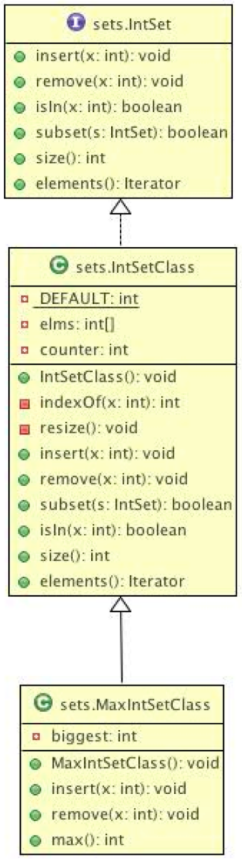

```
  public boolean subset(IntSet s) {
    if (s.size() < this.size()) return false;
    for (int i = 0; i < counter; i++)
      if (!s.isIn(elms[i]))
        return false;
    return true;
  }

  public boolean isIn(int x) {
    return (indexOf(x) != -1);
  }

  public int size(){
    return counter;
  }

  public Iterator elements() {
    return new IteratorClass(elms, counter);
  }
}
```



Especificação de `MaxIntSetClass`


```
public class MaxIntSetClass extends IntSetClass {
  private int biggest;
  public MaxIntSet() {...}
  public void insert(int x) {...}
  public void remove(int x) {...}
  public int max() {...}
}
```

- Repare que:
  - A constante **`DEFAULT`**, por ser de classe, não é herdada
  - As duas variáveis de instância da super-classe, **`elms`** e **`counter`**, são herdadas mas não podem ser acedidas directamente por serem privadas
  - O construtor da super-classe não é herdado
  - O método **`indexOf`** da super-classe não é herdado, por ser privado
  - Temos uma nova variável **`biggest`**
  - Temos um novo construtor **`MaxIntSet`** e um novo método max
  - Temos dois métodos redefinidos **`insert`** e **`remove`**
  - **Esta classe também implementa a interface `IntSet`**

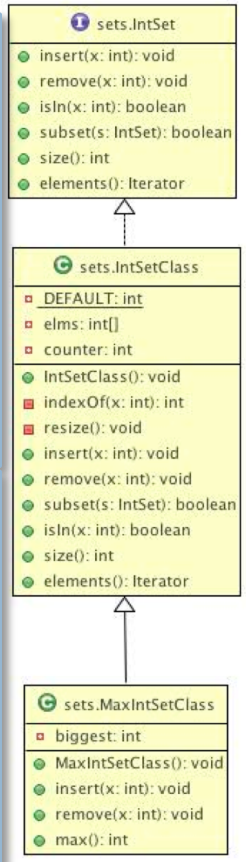


Implementação de `MaxIntSetClass`


```
public class MaxIntSetClass extends IntSetClass {
  private int biggest;

  public MaxIntSetClass() {
    super();
    biggest = 0;
  }

  public void insert(int x) {
    if (size() == 0 || x > biggest)
      biggest = x;

    super.insert(x);
  }
```

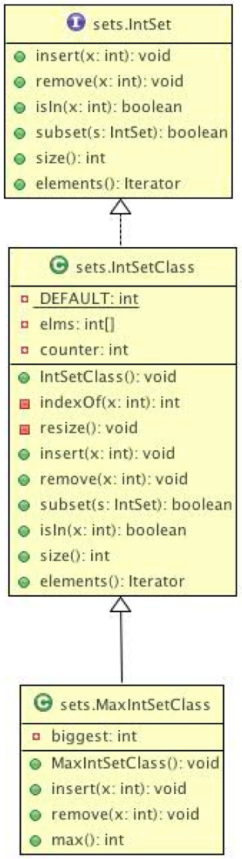
Como em todos os construtores devemos inicializar todas as variáveis de instância. Para garantir que as variáveis inacessíveis da super-classe também são inicializadas, usa-se a chamada ao construtor da super-classe!

O insert começa por tratar do caso especial levantado pela necessidade de actualizar a variável biggest. No resto, seria igual ao da super-classe, portanto, delega nela a implementação.



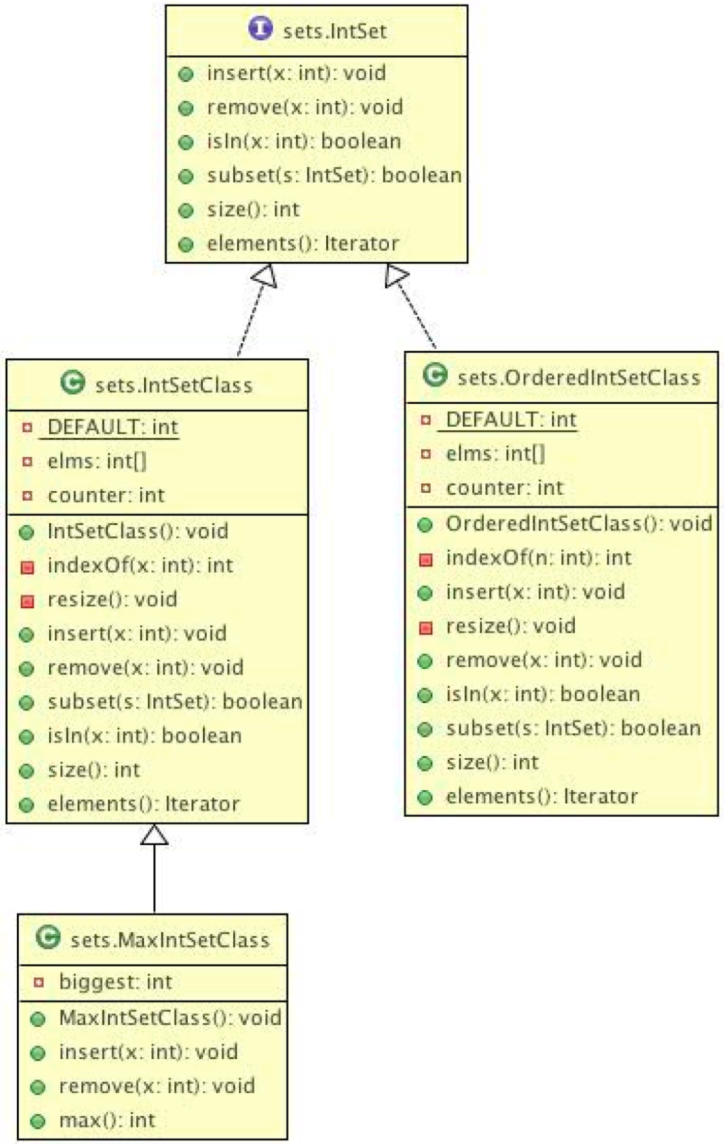
Implementação de `MaxIntSetClass`


```
  public void remove(int x) {
    super.remove(x);

    if ((size()>0) && (x == biggest)) {
      Iterator it = elements();
      biggest = it.next();
      int tmp;
      while (it.hasNext()) {
        tmp = it.next();
        if (tmp > biggest)
          biggest = tmp;
      }
    }
  }

  public int max() { return biggest; }
}
```



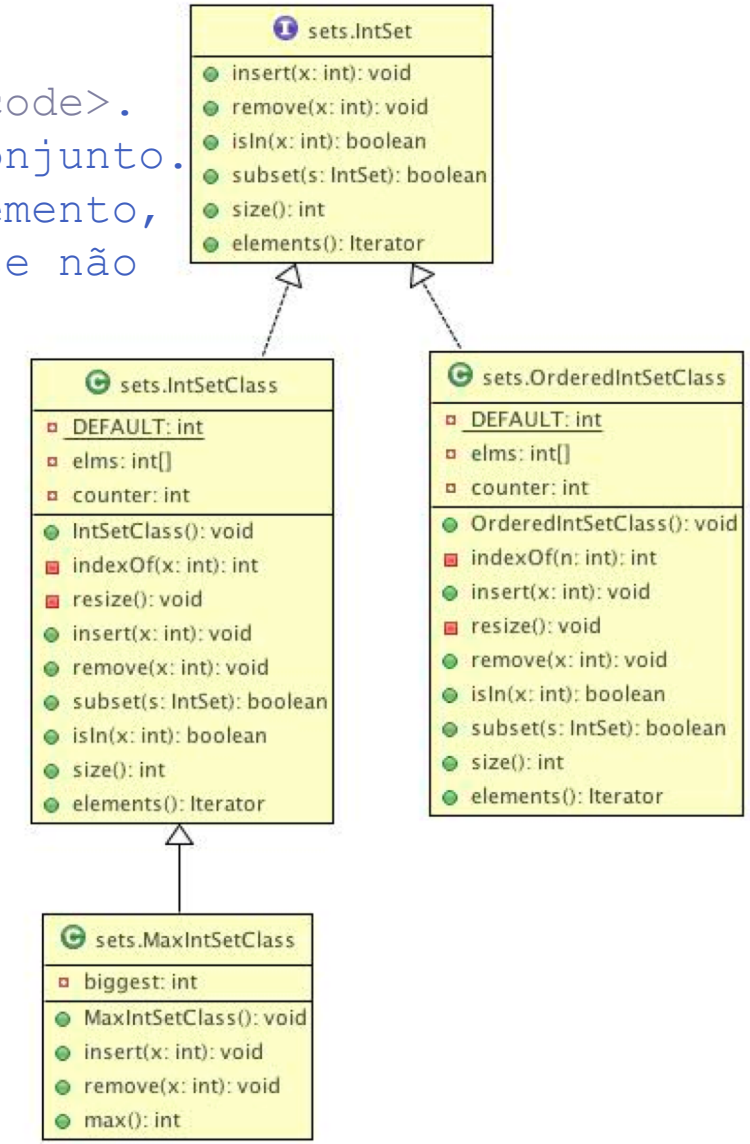
Implementação de `OrderedIntSetClass`


```
public class OrderedIntSetClass
                        implements IntSet {
  private static final int DEFAULT = 10;
  private int[] elms;
  private int counter;

  /**
   * Inicializa o vector acompanhado,
   * de modo a representar
   * um conjunto vazio de inteiros.
   */
  public OrderedIntSetClass(){
    elms = new int[DEFAULT];
    counter = 0;
  }
```



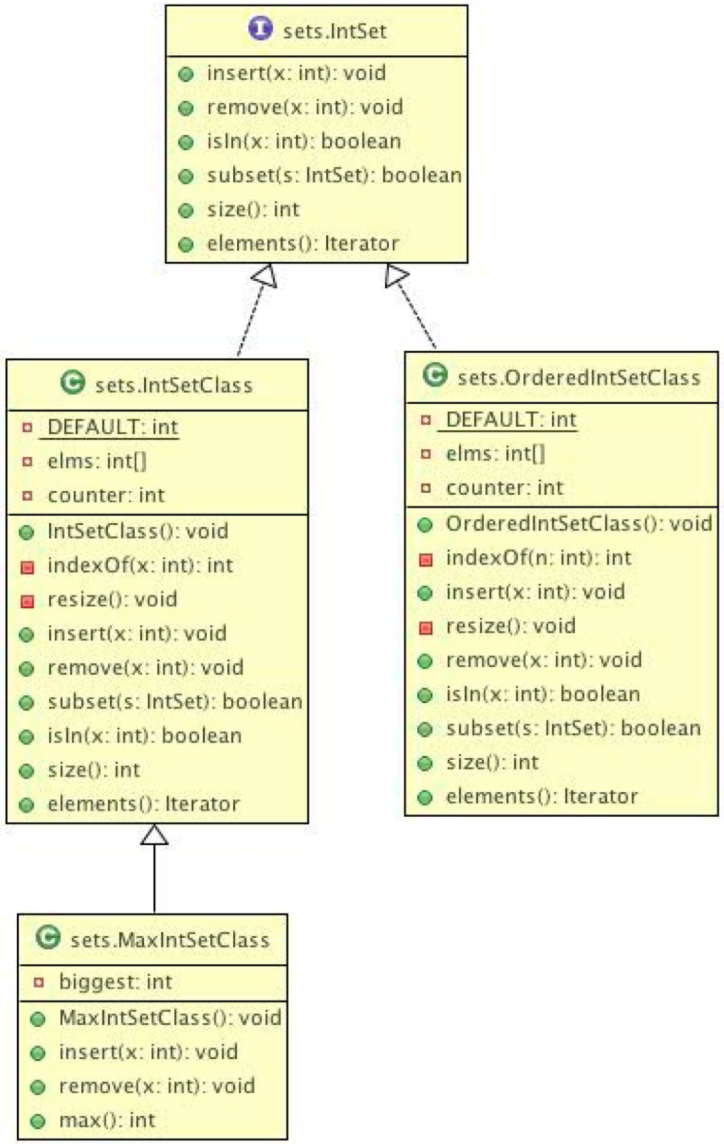
Implementação de `OrderedIntSetClass`


```
/**
 * Devolve o índice do elemento <code>n</code>.
 * @param n - o elemento a pesquisar no conjunto.
 * @return - o índice com a posição do elemento,
 * ou o índice do primeiro inteiro maior se não
 * existir.
 */
private int indexOf(int n) {
  int low = 0;
  int high = counter-1;
  int mid = -1;
  while (low <= high) {
    mid = (low+high)/2;
    if (elms[mid] == n) return mid;
    else if (n < elms[mid]) high = mid-1;
    else low = mid+1;
  }
  return low;
}
```



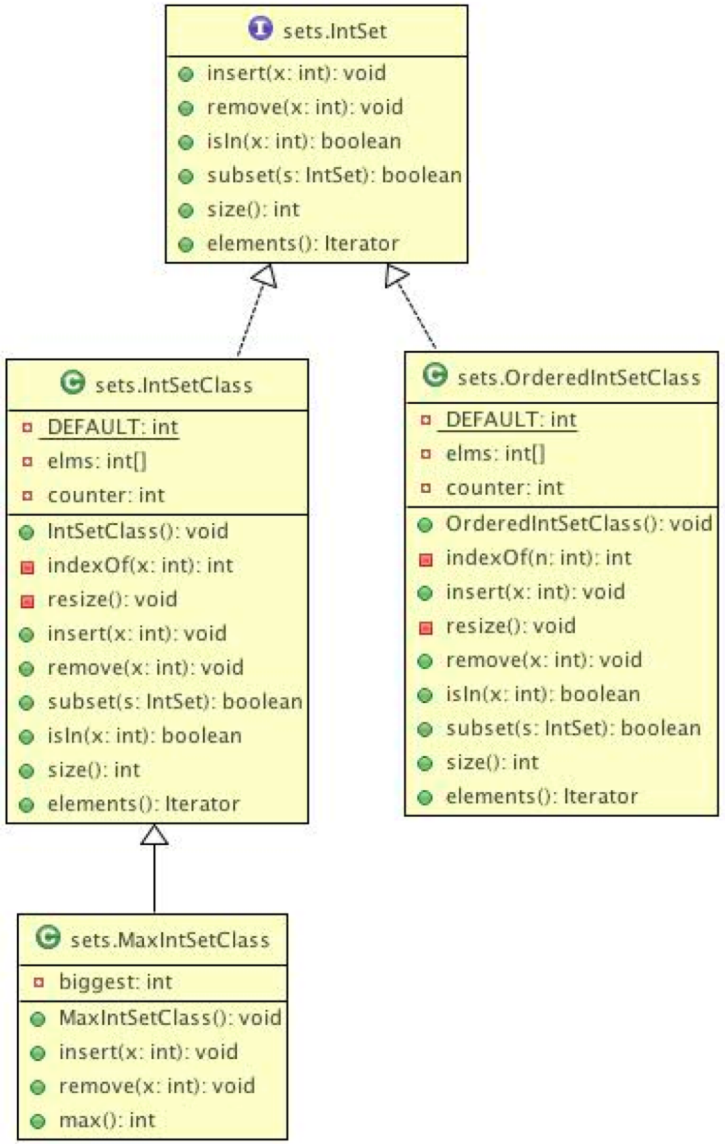
Implementação de `OrderedIntSetClass`


```
public void insert(int x) {
  int pos = indexOf(x);
  if (counter == elms.length)
    resize();
  for (int i = counter; i > pos; i--)
    elms[i] = elms[i-1];
  elms[pos] = x;
  counter++;
}

private void resize() {
  int[] tmp = new int[elms.length*2];
  for (int i = 0; i < counter; i++)
      tmp[i] = elms[i];
  elms = tmp;
}
```



Implementação de `OrderedIntSetClass`


```
public void remove(int x) {
  int i = indexOf(x);
  while (i < counter-1) {
    elms[i] = elms[i+1];
    i++;
  }
  counter--;
}

public boolean isIn(int x) {
  int i = indexOf(x);
  if (counter == i)
    return false;
  return elms[i] == x;
}
```



Implementação de `OrderedIntSetClass`


```
  public boolean subset(IntSet s) {
    if (s.size() < this.size())
      return false;
    for (int i = 0; i < counter; i++)
      if (!s.isIn(elms[i]))
        return false;
    return true;
  }

  public int size() {
    return counter;
  }

  public Iterator elements() {
    return new IteratorClass(elms, counter);
  }
}
```