Igualdade de objectos

- Como implementar os métodos
  - `public boolean existeAluno(Aluno a)`
  - `public void removeAluno(Aluno a)`
- como é que determinamos se o objecto está efectivamente dentro do array de alunos?

- A solução
- `alunos[i] == a`, não é eficaz porque compara os apontadores (e pode ter havido previamente um clone)
  - `(alunos[i]).getNumero() == a.getNumero()`, assume demasiado sobre a forma como se comparam alunos
- Quem é a melhor entidade para determinar como é que se comparam objectos do tipo Aluno?

- através da disponibilização de um método, na classe Aluno, que permita comparar instâncias de alunos
  - é importante que esse método seja universal, isto é, que tenha sempre a mesma assinatura
  - é importante que todos os objectos respondam a este método
- **public boolean equals(Object o)**

- dessa forma o método `existeAluno(Aluno a)` da classe Turma, assume a seguinte forma:

```
public boolean existeAluno(Aluno umAluno) {
  boolean resultado = false;

  if (umAluno != null) {

    for(int i=0; i< this.ocupacao && !resultado; i++)
      resultado = this.alunos[i].equals(umAluno);

    return resultado;
  }
  else
    return false;
}
```

- Em resumo:
  - o método de igualdade é determinante para que seja possível ter colecções de objectos
  - o método de igualdade entre objectos de uma classe não pode ser codificado a não ser pela classe: abstração de dados
  - existe um conjunto de regras básicas que todos os métodos de igualdade devem respeitar

O método equals

- a assinatura é:

```
public boolean equals(Object o)
```

- é importante referir, antes de explicar em detalhe o método, que:

O método equals

- a relação de equivalência que o método implementa:
- é **reflexiva**, ou seja x.equals(x) == true, para qualquer valor de x que não seja nulo
- é **simétrica**, para valores não nulos de x e y se x.equals(y) == true, então y.equals(x) == true

- é **transitiva**, em que para x,y e z, não nulos, se x.equals(y) == true, y.equals(z) == true, então x.equals(z) == true

- é **consistente**, dado que para x e y não nulos, sucessivas invocações do método equals (x.equals(y) ou y.equals(x)) dá sempre o mesmo resultado

- para valores nulos, a comparação com x, não nulo, dá como resultado false.

- quando os objectos envolvidos sejam o mesmo, o resultado é true, ie, x.equals(y) == true, se x == y
  - dois objectos são iguais se forem o mesmo, ie, se tiverem o mesmo apontador
- caso não se implemente o método equals, temos uma implementação, por omissão, com o seguinte código:

```
public boolean equals(Object object) {
        return this == object;
}
```

- template típico de um método equals

```
public boolean equals(Object o) {

    if (this == o)
      return true;

    if((o == null) || (this.getClass() != o.getClass()))
      return false;

    <CLASSE> m = (<CLASSE>) o;
    return ( <condições de igualdade> );

}
```

- o método equals da classe Aluno

```java
/**
* Implementação do método de igualdade entre dois Aluno
* Redefinição do método equals de Object.
*
* @param  umAluno   aluno que é comparado com o receptor
* @return      booleano true ou false
*/
public boolean equals(Object o) {
  if (this == o)
    return true;

  if((o == null) || (this.getClass() != o.getClass()))
    return false;

  Aluno umAluno = (Aluno) o;
  return(this.nome.equals(umAluno.getNome()) && this.nota == umAluno.getNota()
        && this.numero.equals(umAluno.getNumero())
        && this.curso.equals(umAluno.getCurso()));
}
```

- como é que será o método equals da classe Turma?

- quais as consequências de não ter o método equals implementado??
  - consideremos que Aluno "não tem" equals (tem apenas o que é fornecido por omissão)
  - o que acontece neste método de Turma?

```
public boolean existeAluno(Aluno umAluno) {
  boolean resultado = false;

  if (umAluno != null) {

    for(int i=0; i< this.ocupacao && !resultado; i++)
      resultado = this.alunos[i].equals(umAluno);

    return resultado;
  }
  else
    return false;
}
```

O método toString

- a informação deve ser concisa (sem *acuçar de ecran*), mas ilustrativa
- todas as classes devem implementar este método
- caso não seja implementado a resposta será:

getClass().getName() + '@' + Integer.toHexString(hashCode())

O método toString

- implementação *normal* de toString na classe Aluno

```
/**
 * Implementação do método toString
 * comum na maioria das classes Java
 *
 * @return     uma string com a informação textual do objecto aluno
 */
public String toString() {

    return("Numero:" + this.numero + "Nome:"+ this.nome + "Nota:" + this.nota);

}
```

- o operador "+" é a concatenação de Strings, sempre que o resultado seja uma String

- Strings são objectos imutáveis, logo não crescem, o que as torna muito ineficientes
  - Para tornar a construção de Strings mais simples (e legível) pode recorrer-se à utilização da classe StringBuilder

```
/**
 * Implementação do método toString
 * comum na maioria das classes Java
 *
 * @return     uma string com a informação textual do objecto aluno
 */
public String toString() {
    StringBuilder sb= new StringBuilder();

    sb.append("Numero: ");
    sb.append(this.numero+"\n");
    sb.append("Nome: ");
    sb.append(this.nome+"\n");
    sb.append("Nota: ");
    sb.append(this.nota+"\n");

    return sb.toString();
}
```

...completar a classe Turma

- equals

```java
/**
 * Método equals standard do Java.
 * Utiliza o método privado getAlunos para efectuar a comparação entre
 * duas instâncias de turma.
 */

public boolean equals(Object umaTurma) {
  if (this == umaTurma)
    return true;

  if((umaTurma == null) || (this.getClass() != umaTurma.getClass()))
    return false;
  else {
    Turma turma = (Turma) umaTurma;
    return (this.designacao.equals(turma.getDesignacao())
           && this.capacidade == turma.getCapacidade()
           && this.ocupacao == turma.getOcupacao()
           && Arrays.equals(this.alunos,turma.getAlunos()));

  }
}
```

- nesta versão recorreu-se ao método equals da classe Arrays
- é necessário garantir que a remoção de alunos não deixa "lixo" no *array* alunos

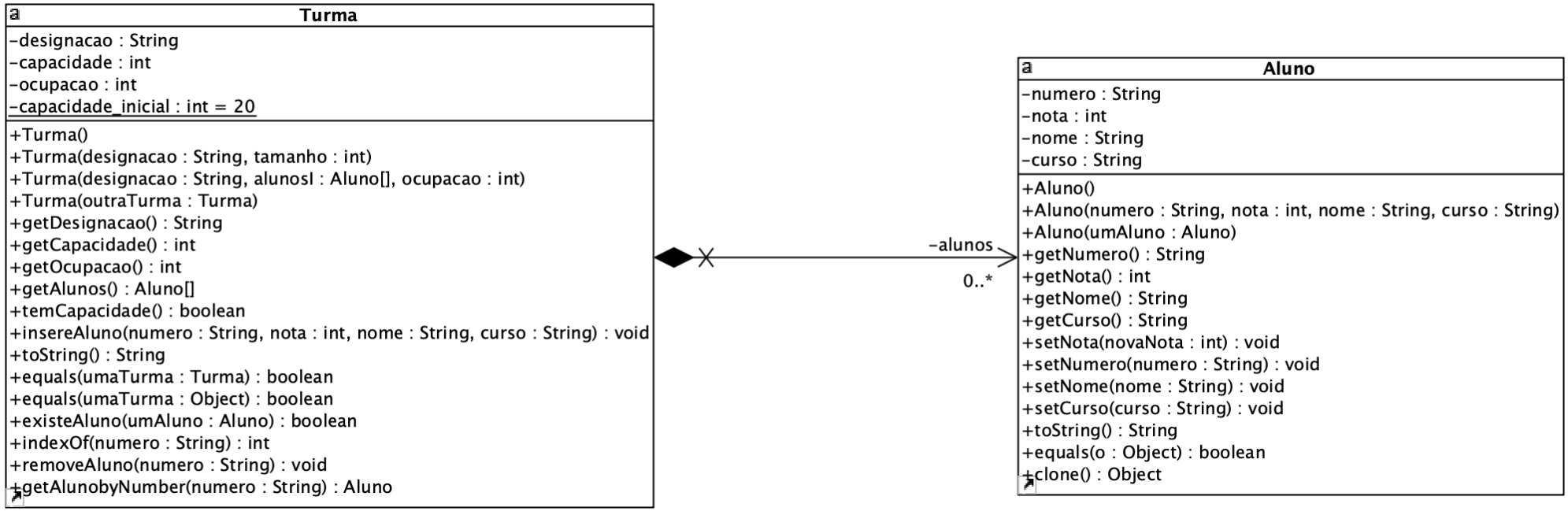
- toString

```
/**
 * Método toString por questões de compatibilização com as restantes
 * classes do Java.
 *
 * Como o toString é estrutural e a classe Aluno tem esse método
 * implementado o resultado é o esperado.
 *
 */
public String toString() {
  StringBuffer sb = new StringBuffer();

  sb.append("Designação: "); sb.append(this.designacao+"\n");
  sb.append("Capacidade: "); sb.append(this.capacidade+"\n");
  sb.append("Alunos: "+"\n"); sb.append(this.alunos.toString());

  return sb.toString();
}
```

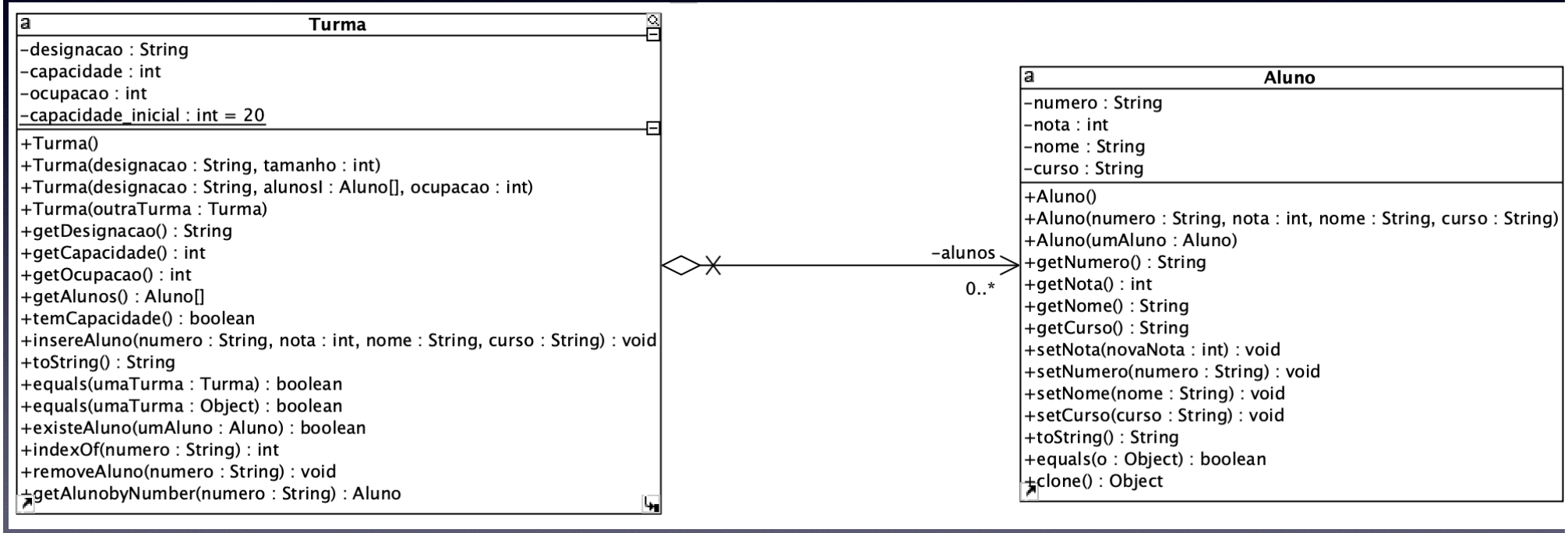
- clone

```
public Turma clone() {
  return new Turma(this);
}
```

A arquitectura com composição



A arquitectura com agregação



… em resumo

- Se o diagrama de classes indicar uma associação de **composição**:
  - faz-se uma cópia (clone) dos objectos quando são guardados internamente
  - devolve-se sempre uma cópia dos objectos e, caso seja necessário, da estrutura de dados que os guarda

… em resumo

- Se o diagrama de classes indicar uma associação de **agregação**:
  - guarda-se internamente o apontador dos objectos passados como parâmetro
  - devolve-se sempre o apontador dos objectos e, caso seja solicitado, uma cópia da estrutura de dados que os guarda

… em resumo

- Quando o diagrama de classes não explicitar se a associação é de composição ou de agregação, parte-se do princípio que é de **composição**!
- O mesmo se aplica quando não se fornece o diagrama de classes.