Caso de Estudio: TDA Racional

Un **tipo de dato** es un modelo matemático que especifica un conjunto de valores y un conjunto de operaciones sobre esos valores.

El **encapsulamiento** es un mecanismo que permite dividir los programas en módulos que pueden ser usados conociendo sólo su interfaz.

Un tipo de dato es **abstracto** si la representación de su estado interno y la implementación de sus operaciones queda encapsulada.

La clase **`Racional`** define un tipo de dato abstracto a partir del cual es posible crear instancias.

Caso de Estudio: TDA Racional

Antes de usar el TDA Racional en una aplicación específica debemos verificar que cumple con lo especificado.

Podemos probar las operaciones en forma interactiva o definir una clase Tester que use a la clase Racional para verificar sus servicios.

El mismo problema podría diseñarse de manera diferente, sin modificar la interface ni la funcionalidad de los servicios. En este caso el mismo tester permite verificar las dos implementaciones.

Caso de Estudio: TDA Racional

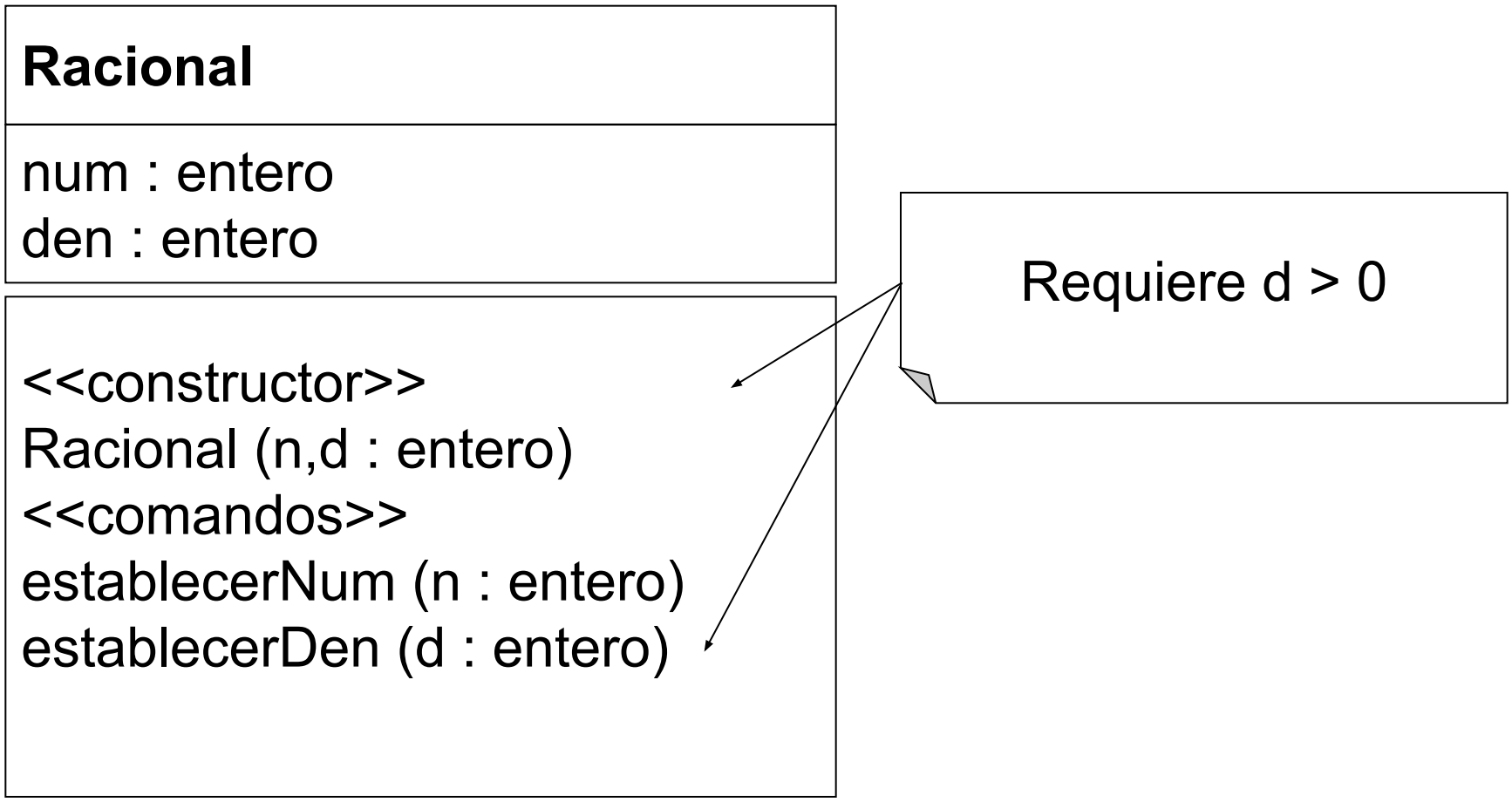
*Diseñe e implemente un **tipo de dato abstracto** que permita representar números racionales.*

Racional
num : entero den : entero

Una alternativa es representar a cada número racional como un par, en donde el primer elemento es el numerador y el segundo el denominador.

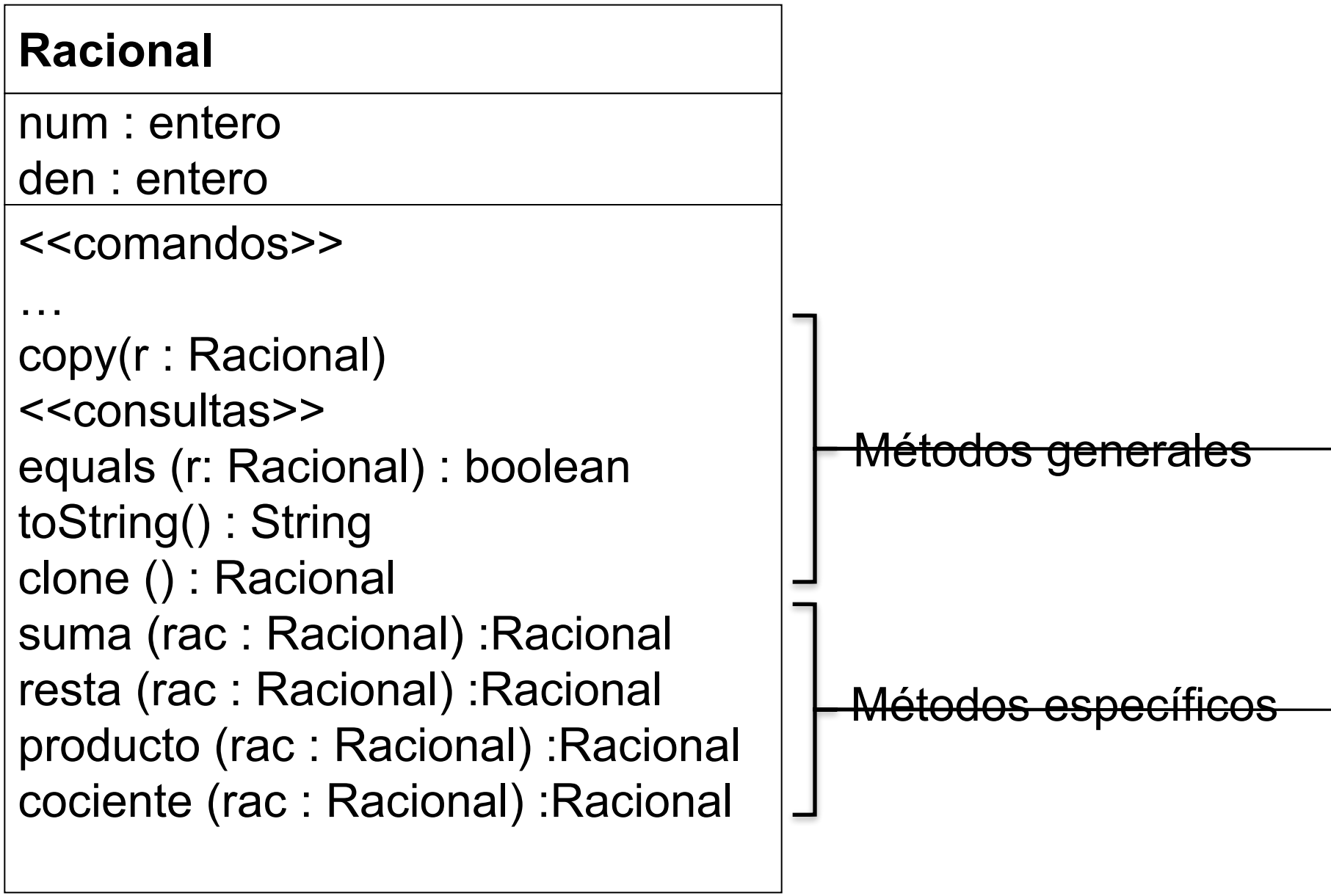
¿Todo número racional puede representarse de esta manera? ¿Todo par de enteros representan a un número racional?

Caso de Estudio: TDA Racional



El comentario asociado al constructor establece la **responsabilidad de la clase cliente** de controlar el valor de un atributo antes de crear un racional o modificar su denominador

Caso de Estudio: TDA Racional



Caso de Estudio: TDA Racional

```
class Racional {

private int num;
private int den;

// CONSTRUCTOR
public Racional(int n, int d ) {
//Requiere d > 0
    num = n;
    den = d;
}
```

Caso de Estudio: TDA Racional

```
// COMANDOS
public void establecerNum(int n ) {
    num = n;
}
public void establecerDen(int d ) {
//Requiere d > 0
    den = d;
}
```

Caso de Estudio: TDA Racional

```
// COMANDOS
…
public void copy(Racional r ) {
   num  = r.obtenerNum();
   den = r.obtenerDen();
}
```

Caso de Estudio: TDA Racional

```
// CONSULTAS
public int obtenerNum() {
   return num;
}
public int obtenerDen() {
   return den;
}
```

Caso de Estudio: TDA Racional

```
// CONSULTAS
public boolean equals(Racional r){
    int rnum = r.obtenerNum();
    int rden = r.obtenerDen();
    int rmcd = mcd (rnum,rden);
    int MCD = mcd (num,den)
    return num/MCD == rnum/rmcd &&
           den/MCD == rden/rmcd;
}
```

```
private int mcd(int a, int b) {
  int r;
  while (a % b !=0){
    r = a % b;
    a = b;
    b = r;
  }
  return b;
}
```

Caso de Estudio: TDA Racional

```
// CONSULTAS
…
public String toString() {
   return ( num + "/" + den );
}
public Racional clone  (){
   Racional r = new Racional(num,den);
   return r;
}
```

```
public Racional clone  (){
   return new Racional(num,den);
}
```

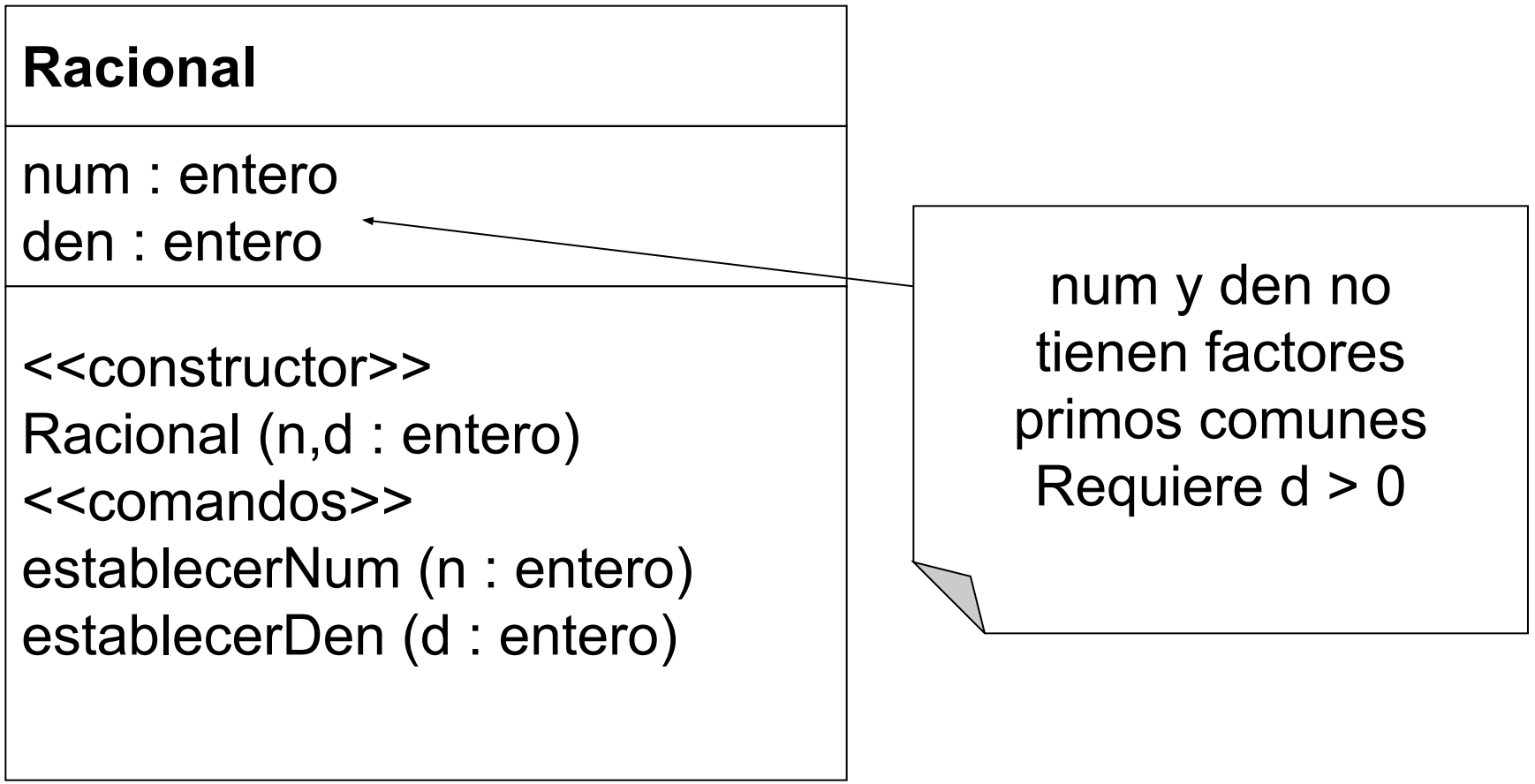
Caso de Estudio: TDA Racional

```
public Racional suma( Racional op ) {
  int n = num*op.obtenerDen()
              +den*op.obtenerNum();
  int d = den * op.obtenerDen();
  return( new Racional(n, d ) );
    }
public Racional resta( Racional op ) {
  int n = num*op.obtenerDen()
              - den*op.obtenerNum();
  int d = den * op.obtenerDen();
     Racional s = new Racional(n,d);
  return s;
    }
```

Caso de Estudio: TDA Racional

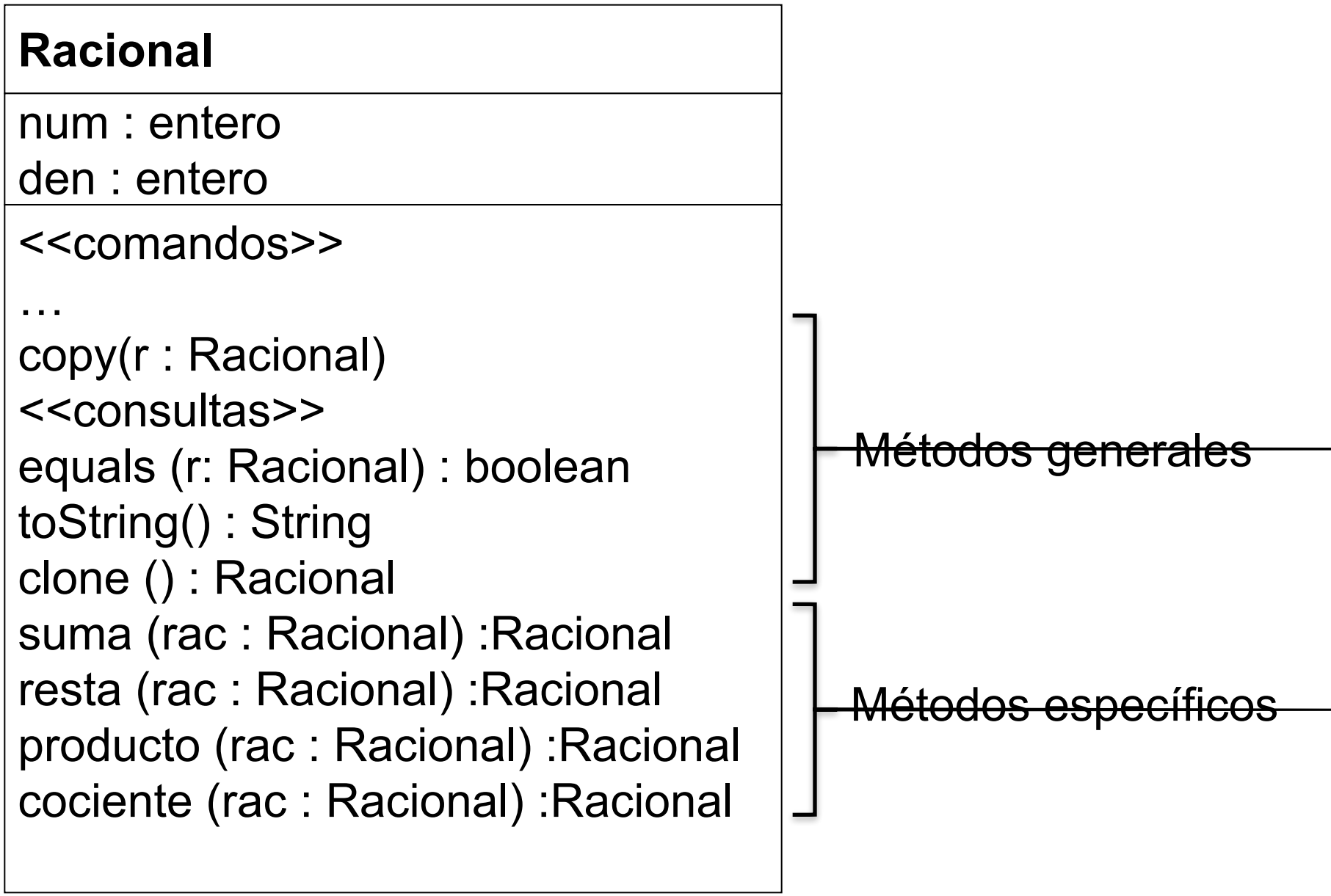
```
public Racional producto( Racional op ) {
   return
       new Racional(num*op.obtenerNum(),
          den*op.obtenerDen());
     }

public Racional cociente( Racional op){
   return
    new Racional (num*op.obtenerDen(),
        den*op.obtenerNum());
}
```

Caso de Estudio: TDA Racional



El comentario asociado al constructor establece un **compromiso respecto a la representación** de un racional que afecta a los comandos

Caso de Estudio: TDA Racional



Caso de Estudio: TDA Racional

```
class Racional {

private int num;
private int den;
/*num y den no tienen factores primos
comunes*/
// CONSTRUCTOR
public Racional(int n, int d ) {
//Requiere d > 0
    int MCD = mcd (n,d);
      num = n/MCD;
    den = d/MCD;
}
```

Caso de Estudio: TDA Racional

```
  // COMANDOS
public void establecerNum(int n ) {
   int MCD = mcd (n,den);
   num = n/MCD;
   den = den/MCD;
}
public void establecerDen(int d ) {
   int MCD = mcd (num,d);
   num = num/MCD;
   den = d/MCD;
}
```

Caso de Estudio: TDA Racional

```
// COMANDOS
…
public void copy(Racional r ) {
   num = r.obtenerNum();
   den = r.obtenerDen();
}
```

Caso de Estudio: TDA Racional

```
// CONSULTAS
public int obtenerNum() {
   return num;
}
public int obtenerDen() {
   return den;
}
```

Caso de Estudio: TDA Racional

```
// CONSULTAS
public boolean equals(Racional r){
   return num == r.obtenerNum()&&
      den == r.obtenerDen();
}
public String toString() {
   return ( num + "/" + den );
}
public Racional clone  (){
   Racional r= new Racional(num,den);
      return r;
}
```