Sesión 1. Números racionales

Crea un proyecto Java de nombre, `sesión-01`, y cambia las propiedades de éste abriendo el menú contextual (haz clic con el botón derecho del ratón sobre el proyecto) y cambia la codificación de caracteres a UTF-8 (los archivos que se proporcionen para las prácticas siempre utilizaran este código de caracteres). Otra opción es cambiar la codificación de caracteres del *workspace* a UTF-8 antes de crear el proyecto, ya que por defecto los proyectos se crean con la codificación de caracteres del espacio de trabajo. Para hacer esto hay que cambiar las preferencias del Eclipse y expandir *General*, las preferencias del *Workspace* están al final del árbol *General*.

En todos los proyectos Java para prácticas se crearán, al menos, dos *packages*: *estDatos* y *app*. El *package estDatos* contendrá el código de las abstracciones que se requieran (funcionales, de datos o iterativas) y el *package app* contendrá el código de los programas cliente (programas que utilizan las abstracciones). De esta forma también se independiza la implementación de una abstracción de su uso desde el punto de vista del lenguaje utilizado (Java).

En esta primera práctica se implementará, al menos, un tipo de dato para números racionales y se hará alguna modificación al TDA Racional (interfaz) que se proporciona.

Material proporcionado para la práctica (descarga del CV y descomprime el archivo *sesión-01.zip*):

1. La biblioteca *pair.jar* que contine los archivos compilados (`.class`) del TDA *Pair<K, V>* de pares de elementos que se utiliza en las diapositivas de teoría para exponer algunos conceptos:
    a. La interfaz *Pair<K, V>*
    b. La clase abstracta *AbstractPair<K, V>* que implementa parcialmente la interfaz previa para proporcionar las operaciones *equals* y *toString* para los pares
    c. Dos implementaciones concretas, los tipos de datos *PairImp<K, V>* y *PairImpAlt<K, V>*. Para la práctica que se solicita se puede utilizar una cualquiera de ambas, ya que la única diferencia entre éstas es la forma de representar los pares y esto es irrelevante para su uso.
2. Una carpeta de nombre *doc* con la documentación de la interfaz y clases citadas en el apartado previo. Para acceder a la documentación abrir el archivo *index.html*.
3. Dos carpetas de nombre *app* y *estDatos* con archivos fuente de Java que deberás copiar a los *packages* del mismo nombre. En la carpeta *app* puedes encontrar un programa de ejemplo de uso de racionales al que puedes añadir más código para probar las operaciones de tus racionales. La carpeta *estDatos* contiene el TDA *Rational* (nótese que el TDA establece que el signo del número racional será el signo de su numerador; es decir, que el denominador siempre debe quedar como positivo).

Se pide:

1. Crear una clase abstracta de nombre *AbstractRational* que implemente parcialmente el TDA *Rational*. Inicialmente esta clase únicamente proporcionará la operación *toString* para números racionales. La cadena de caracteres que representará a un racional deberá ser la habitual; es decir, n/d siendo n su numerador y d su denominador, excepto si $d=1$, en cuyo caso simplemente será n.
2. Implementar el *tipo de dato inmutable* (o *no modificable*) *InmutableRational*, extendiendo la clase abstracta del apartado anterior y almacenando la información en un par (un campo de tipo *Pair<K, V>*). Este tipo de dato deberá incluir también un *constructor de conversión* (*InmutableRational(Rational r)*).
3. Dado que los números racionales se pueden ordenar (tienen *orden natural*), modificar la interfaz *Rational* para que extienda la interfaz *Comparable<Rational>* (recuérdese que la interfaz es *Comparable<T>* y, en

este caso, lo que se van a comparar son números racionales; es decir, T no es cualquier tipo, tiene que ser *Rational*). Implementa la operación *compareTo* donde consideres más conveniente.

Nota: para disponer de los pares en el proyecto deberás abrir el cuadro de diálogo ***Java Build Path***. Éste se puede obtener abriendo el submenú `Build Path` del menú contextual del proyecto para seleccionar la opción `Configure Build Path…` Una vez abierto el cuadro de diálogo deberás seleccionar la opción `Java Build Path` en el listado de opciones de la izquierda y, posteriormente, abrir la pestaña `Libraries`. Por último, añade la biblioteca proporcionada, *pair.jar*, como *jar* externo y cierra el cuadro de diálogo.

Sobre las operaciones heredadas de *Object*

Lo más habitual es que se redefinan para cualquier tipo de dato, al menos las operaciones: *toString*, *equals* y *hashCode*.

Normalmente los IDEs de Java generan de forma automática un esqueleto de estas operaciones y Eclipse no es una excepción. Los esqueletos proporcionados deberán adaptarse para el tipo de dato concreto que se esté implementando, sobre todo la operación *toString*, a pesar de lo cual no es raro que estas implementaciones nos deparen alguna desagradable sorpresa. Dicho de otra forma, la adaptación requerida puede resultar más compleja de lo que inicialmente pudiera parecer.

Por otra parte, las operaciones *equals* y *hashCode* están relacionadas, para dos objetos x e y cualesquiera debe cumplirse:

$$x.equals(y) \rightarrow x.hashCode() = y.hashCode()$$

y, para mejorar el rendimiento de los diccionarios (éstos se verán en el tema 5), se recomienda, aunque no es obligatorio, que:

$$\neg x.equals(y) \rightarrow x.hashCode() \neq y.hashCode()$$

por eso los IDEs generan ambas operaciones de forma simultánea.

Operaciones generadas por el IDE Eclipse

```
@Override
public int hashCode() {
    final int prime = 31;
    int result = super.hashCode();
    result = prime * result + Objects.hash(this.denominator(), this.numerator());
    return result;
}

@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (!super.equals(obj))
        return false;
    if (!(obj instanceof Rational))
        return false;
    Rational other = (Rational) obj;
    return this.denominator() == other.denominator() &&
            this.numerator() == other.numerator();
}
```

A modo de ejemplo, añade a la clase abstracta las operaciones *equals* y *hashCode* indicadas previamente. No se pide que se generen de forma automática porque sólo se pueden generar a partir de campos de datos y la clase *AbstractRational* carece de ellos. Pero las implementaciones que se proporcionan son las que genera el Eclipse para un racional que directamente esté representado por dos enteros: su numerador y su

denominador. Por razones obvias, el único cambio que se ha realizado a la implementación generada por el Eclipse ha sido cambiar los campos correspondientes al numerador y denominador por las llamadas a las operaciones *numerator()* y *denominator()*, respectivamente. Además, el tipo al que se convierte el objeto que recibe como argumento la operación *equals* se ha cambiado a *Rational* (en el código generado se convierte a la clase que implementa el TDA *Rational*).

Declara ahora en el programa de ejemplo dos racionales, *rat1* y *rat2*, de valores 2/3 y 4/6, respectivamente y muestra en consola los resultados de las invocaciones: *rat1.equals(rat2)*, *rat1.hashCode()* y *rat2.hashCode()*. ¿Por qué se producen los errores y cómo habría que implementar las operaciones para que ambas sean correctas?

Sugerencia: consulta las especificaciones de las operaciones *equals* y *hashCode* en la clase *Object*