UT6_PD7

Property	COMPLETED
Date	@June 5, 2022
BLOCKED	

Ejercicio 1:

Implementacion en Object:

```
/**
     * Returns a hash code value for the object. This method is
     * supported for the benefit of hash tables such as those provided by
     * {@link java.util.HashMap}.
     * <p>
     * The general contract of {@code hashCode} is:
     * <ul>
     * <li>Whenever it is invoked on the same object more than once during
     *     an execution of a Java application, the {@code hashCode} method
     *     must consistently return the same integer, provided no information
     *     used in {@code equals} comparisons on the object is modified.
     *     This integer need not remain consistent from one execution of an
     *     application to another execution of the same application.
     * <li>If two objects are equal according to the {@code equals(Object)}
     *     method, then calling the {@code hashCode} method on each of
     *     the two objects must produce the same integer result.
     * <li>It is <em>not</em> required that if two objects are unequal
     *     according to the {@link java.lang.Object#equals(java.lang.Object)}
     *     method, then calling the {@code hashCode} method on each of the
     *     two objects must produce distinct integer results.  However, the
     *     programmer should be aware that producing distinct integer results
     *     for unequal objects may improve the performance of hash tables.
     * </ul>
     * <p>
     * As much as is reasonably practical, the hashCode method defined
     * by class {@code Object} does return distinct integers for
     * distinct objects. (The hashCode may or may not be implemented
     * as some function of an object's memory address at some point
     * in time.)
     *
     * @return  a hash code value for this object.
*/
@HotSpotIntrinsicCandidate
public native int hashCode();
```

Implementacion String:

```
/**
     * Returns a hash code for this string. The hash code for a
     * {@code String} object is computed as
     * <blockquote><pre>
     * s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]
     * </pre></blockquote>
     * using {@code int} arithmetic, where {@code s[i]} is the
     * <i>i</i>th character of the string, {@code n} is the length of
     * the string, and {@code ^} indicates exponentiation.
     * (The hash value of the empty string is zero.)
     *
     * @return  a hash code value for this object.
     */
    public int hashCode() {
        int h = hash;
        if (h == 0 && value.length > 0) {
            hash = h = isLatin1() ? StringLatin1.hashCode(value)
                                  : StringUTF16.hashCode(value);
        }
        return h;
    }
```

Implementacion Integer:

```
/**
     * Returns a hash code for this {@code Integer}.
     *
     * @return  a hash code value for this object, equal to the
     *          primitive {@code int} value represented by this
     *          {@code Integer} object.
     */
public int hashCode() {
        return Integer.hashCode(value);
    }
//---------------------------------------------------
public static int hashCode(int value) {
        return value;
    }
```

📌 Extraido del java jdk

Ejercicio 2:

El Hash designa claves unicas a valores, con la posibilidad de que estos valores sean recuperados al pasarle. HashMap tiene un array de nodos y el nodo esta representado como una clase dentro de la clase mapa. Un array de nodos se llama

Buckets. Al momento de la insercion de un elemento se calcula el hashcode de la clave (con el codigo mostrado a continuacion)

```
index = hashCode(Key) & (n-1)
//where n = array.length
```

La estructura de un HashMap es un arreglo de Nodos<K,V>

“Hola” → Hola.hashCode() & (4-1) = 0

“HolaMundo” → HolaMundo.hashCode() & (4-1) = 1

“HashMap” → HashMap.hashCode() & (4-1) = 2

“Colecciones” → Colecciones.hashCode() & (4-1) = 0

Ejercicio 3:

```
public class Alumnos extends Object {
    private int ID;
    private String fullName;
    private String email;

    @Override
    public int hashCode() {
        return this.ID*this.fullName.hashCode()*this.email.hashCode();
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null) return false;
        if (getClass() != o.getClass()) return false;

        final Alumnos other = (Alumnos) o;

        if (this.ID != other.ID) return false;
        if (!Objects.equals(this.fullName, other.fullName)) return false;

        return Objects.equals(this.email, other.email);
    }
}
```

La idea es que en el hashCode de Alumno se utilizen los hashCode de los diferentes propiedades de las clases ademas de multiplicarlo por el id que al ser un numero es su propio hashCode.