1.00 Clase adicional 11

6/5/2002

Agenda

- Examen final
- Métodos de la clase *Object*
- BP10
  - *CharPair*
  - *Files*
  - *StringTokenizer*
  - *Exceptions*
  - *HashTable*
- Evaluaciones del curso

1.00 - Examen final

- Abarcará todo el trimestre, pero se centrará en los conceptos más recientes
- Miércoles, 22 de mayo: 1:30-4:30 pm en el Johnson Athletic Center
- Traiga su portátil antes del examen final o en el examen final (¡pero llegue temprano!)

La clase `Object`

- Métodos de la clase **`Object`**:

public boolean equals (Object obj)	Indica si hay otro objeto igual que éste.
public String toString()	Devuelve una representación de la cadena del objeto.
public int hashCode()	Devuelve un valor de código *hash* para el objeto.

El método equals()

```
public class Student {
        int id;
        public Student(int id){
             this.id = id;
        }
        public static void main(String args[]){
              Student s1 = new Student(12);  //el id del estudiante es 12
              Student s2 = new Student (12); //el id del estudiante 12
              Student s3 = s1;
               s1.equals(s2);  //¿qué devolverá el método equals() predeterminado?
               s1.equals(s3);  //¿qué devolverá el método equals() predeterminado?
        }
      }
```

PREGUNTA: ¿Cómo consigue que el equals() devuelva *true* si dos objetos Student tiene en mismo ID? (Pista: ignore el método equals() predeterminado.)

El método equals() (cont.)

```
public class Student {
        int id;
        public Student(int id){
             this.id = id;
        }
        public boolean equals (Object obj) {
          if (obj != null && obj instanceof Student)
               return  id == ((Student) obj).id;
           else
               return false;
      }
        public static void main(String args[ ]){
              Student s1 = new Student(12);  //el id del estudiante es 12
              Student s2 = new Student (12); //el id del estudiante es 12
              Student s3 = s1;
               s1.equals(s2);  //¿qué devuelve ahora?
               s1.equals(s3);  //¿qué devuelve ahora?
       }
      }
```

El método toString()

```
public class Student {
  int id;
  public Student(int id){
        this.id = id;
  }
  //ignoramos el método toString() predeterminado
  public String toString() { return new String ("ID: " + id);}
  public static void main(String args[]) {
        Student s = new Student(12);
        System.out.println (s);                    //ID: 12
        System.out.println ("Estudiante " + s); //ID de estudiante: 12”
        System.out.println (s.toString());     //ID: 12
   }
}
```

BP10 - Problema 1

- Clase CharPair:
  - Constructor
  - toString(): devuelve una cadena adecuada para imprimir
  - equals(): devuelve *true* si los datos de CharPairs son iguales. [Nota: so dos objetos a y b son iguales (a.equals(b) devuelve *true*), sus *hashCode* deben tener el mismo valor.]
  - hashCode(): devuelve una distribución razonablemente uniforme de enteros basados en los datos de caracteres

BP10 - Problema 1

- Cree un archivo de texto “indata.dat”. Algo parecido a:

```
3 50
2.718 3.141
GOOD
Ésta es la primera cadena
Ésta es la segunda cadena
```

- Declare *File* y *BufferedReader* tal como se muestra en el problema 1.
- Lea cada línea utilizando readLine() y utilice *StringTokenizer* su hay más de un elemento en la línea.
- Cree objetos *CharPair*
  CharPair cp1 = new CharPair(‘A’, ‘Z’);
  CharPair cp2 = new CharPair(‘1’, ‘9’);

BP10 - Problema 1

- Datos de salida en pantalla (ejemplo)

```
Read Integers: 3 50
Read Doubles: 2.718 3.141
Read chars: [G] [O] [O] [D]
Read strings: "Ésta es la primera cadena" "Ésta es la segunda cadena"
Hashcode for String:   -1847644   (value: "Ésta es la primera cadena")
Hashcode for Integer:  3   (value: 3)
Hashcode for Double:   -2001631442   (value: 2.718)
Hashcode for CharPair: 2293770   (value: AZ)

CharPair.equals() test: AZ.equals("19") returns false
double1 ^ integer1 = 20.079290231999998
integer2 ^ double2 = 7.135046699010218E24

CP AZ
```

- Aparecen nuevos *Double*, una de las *String* y un nuevo *CharPairs* en el archivo.

BP10 - Problema 1

- Estamos creando objetos `Integer` y `Double` (no primitivos `int` y `double`)
- Utilice constructores que acepten *String*:
- public Integer(String s);
- public Double(String s);
- Obtenga las *String* utilizando *StringTokenizer*
- Asegúrese de cerrar el archivo con file.close()

StringTokenizer (1)

```
import java.util.StringTokenizer;
...
String s = "3.1415926 2.7182818";
StringTokenizer st = new StringTokenizer(s);
String piString = st.nextToken(); // "3.14"
Double d1 = new Double(piString);

String eString = st.nextToken(); // "2.718"
Double d2 = new Double(eString);
```

StringTokenizer (2)

```
import java.util.StringTokenizer;
...
String s = "esto es una cadena";
StringTokenizer st = new StringTokenizer(s);

while (st.hasNext()) {
  String oneToken = st.nextToken();
  System.out.println(oneToken);
}
// ¿qué imprimirá este código?
```

BP10 - Excepciones (prob. 1)

```
try {
  // abre indata.dat, lee datos, etc.
  // cierra indata.dat
  // abre outdata.dat, escribe datos
  // cierra outdata.dat
} catch (IOException e) {
  System.out.println("Excepción: "
   +e.getMessage());
  e.printStackTrace()
}
```

BP10 - Problema 2 (corrector)

dictionary.dat

```
the
of
to
and
a
in
is
it
you
that
he
...
```

key, value identical

dictionary2.dat

```
miniscule     minuscule
minascule     minuscule
millenium     millennium
noticable     noticeable
embarassment  embarrassment
ocurr         occur
ocurred       occurred
occured       occurred
....
```

key is wrong spelling,
value is correct spelling.

BP10 - Problema 2

testfile.dat

```
This is the third and final file, which is to be spell-checked
by your program Teh main point is that you correct occurances of
mispelled words and output
those to a file preserving
line breaks
as they ocurr
in the file
be sure that words ending in s or es are properly spell-checked
it works
it is working
it worked
it fails
it is failing
it failed
Now do the questionaires without harrassing other people
```

BP10 - Problema 2

checked.dat

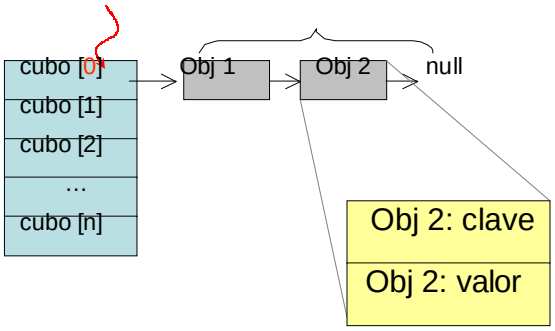
```
this is the THIRD and FINAL FILE, which is to be SPELL-CHECKED
by your PROGRAM the MAIN point is that you CORRECT occurrences
of misspelled words and OUTPUT
THOSE to a FILE
PRESERVING
line BREAKS
as they occur
in the FILE
be SURE that words ending in S or ES are PROPERLY SPELL-CHECKED
it works
it is working
it worked
it FAILS
it is FAILING
it FAILED
now do the questionnaires WITHOUT harassing other people
```

- If a word is misspelled, and you know how to correct it, output correct spelling.
- If a word is misspelled, you don't know how to correct it, output it in ALL CAPS.
- If the word is spelled correctly, output 'as is'.

BP10: *HashTable*

El índice del cubo es el código *hash* de la clave

El contenido de un cubo es una colección de objetos cuyas claves se dispersan al mismo valor (0 en este caso). Puede ser una lista enlazada, un *array*, un vector, un árbol binario, etc.



En general, los objetos de la tabla *hash* tienen dos componentes: una **clave** y un **valor**. El código *hash* se calcula en la *clave*. El *valor* son los datos almacenados. En algunos casos, la clave y el valor coinciden.

BP10: *SimpleMap*

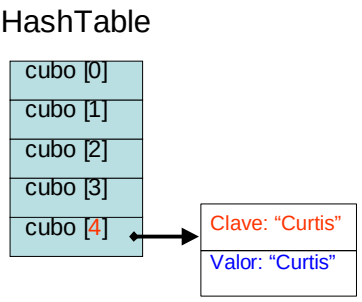
```
public interface SimpleMap {
    public void clear();
    public int dataSize();
    public int tableSize();
    public Object get( Object key );
    public Object remove( Object key );
    public Object put( Object key, Object value );
}
```

P10 - Colocación en la tabla hash

```
HashTable t = new HashTable();
t.put("Curtis", "Curtis")
```

clave valor

1. "Curtis".hashCode() devuelve 2029716094
2. El tamaño de la tabla *hash* (nº de cubos) es 5
3. Índice del cubo = 2029716094 % 5 = 4 (nuestra función "hash2" usa % hashtablesize para asignarlo.)
4. Clave de almacenamiento y par de valores en cubo[4]



BP10: get( ) de *HashTable*

- Invoca get(clave) en *HashTable* para recuperar el valor previamente almacenado.

`get()` devuelve un objeto, por lo que debemos seguir a **`String`**

```
String s=(String)t.get("Curtis");
// s es igual a "Curtis"
```

BP10: pistas para el corrector

- Simplemente cree una clase SpellChecker
- El constructor podrá:
  - Crear HashTable
  - Leer los archivos del diccionario y agregarlos a HashTable (¿cómo lo haría?)
  - Cerrar los archivos
  - Abrir el archivo de texto de entrada (testfile.dat)
  - Abrir el archivo de texto de salida (checked.dat)
  - Leer una línea del archivo de entrada. Para cada línea:
    - Para cada palabra de la línea
      - Corregir ortografía de cada palabra (escribir un método spellCheckWord que devuelva la ortografía correcta del argumento)
      - Mostrar la palabra
    - Mostrar una línea nueva
  - Cerrar todos los archivos

BP10 - spellCheckWord

- Use la documentación de Java para saber más sobre los métodos *String* siguientes:
  - String.length()
  - String.substring()
  - String.toUpperCase()
- ¿Qué diseño utilizará para comprobar las terminaciones (“es”, “s”, “ing”, “ed”)?

Evaluaciones

- J. Harward es el segundo profesor
- Su profesor adjunto es el tercer profesor
- Las preguntas 25-32 están en la siguiente diapositiva
- El estudiante puede llevar, voluntariamente, el formulario cumplimentado al 1-281 o 1-290.

Evaluación: Pregs. 25-32

1=peor, 7=mejor

25. Conceptos fundamentales del desarrollo de software (diseño de programación, desarrollo, depuración)
26. Programación en un entorno interactivo orientado a objetos en Java
27. Uso de la informática en problemas científicos y de ingeniería
28. Interfaces gráficas de usuario (Swing de Java, modelo de eventos)
29. Estructuras de datos (pilas, colas, árboles, listas)
30. Algoritmos (ordenación, búsqueda, recorrido)
31. Patrones de software (modelo-vista-controlador, generador de problemas-solucionador-generador de datos)
32. Uso de los paquetes de Java (matemática incorporada, utilidades, estructuras de datos)