Principios sacados de contexto

¿Todavía programas así?

leanmind.es | carlosble.com

LeanMind
EFFICIENT PEOPLE

~~1. Un solo RETURN!~~

```
public String wordWrap(String text, int width){
  String result;
  if (text == null){
     result = "";
  }
  else if (text.length() <= width){
     result = text;
  }
  else {
     String wrappedLine = text.substring(0, width) + "\n";
     String remainingLines = text.substring(width);
     result = wrappedLine + wordWrap(remainingLines, width);
  }
  return result;
}
```

```
public String wordWrap(String text, int width){
  if (text == null){
     return "";
  }
  if (text.length() <= width){
     return text;
  }
  String wrappedLine = text.substring(0, width) + "\n";
  String remainingLines = text.substring(width);
  return wrappedLine + wordWrap(remainingLines, width);
}
```

LeanMind
EFFICIENT PEOPLE

~~2. Constantes siempre arriba del todo!~~

```
final String LINE_WRAPPER_SYMBOL = "\n";
final String EMPTY_TEXT = "";

/* ...some other methods over here... */

public String wordWrap(String text, int width){
  if (text == null){
     return EMPTY_TEXT;
  }
  if (text.length() <= width){
     return text;
  }
  String wrappedLine = text.substring(0, width) +
                                  LINE_WRAPPER_SYMBOL;
  String remainingLines = text.substring(width);
  return wrappedLine + wordWrap(remainingLines, width);
}
```

```
public String wordWrap(String text, int width){
  String result;
  if (text == null){
     result = "";
  }
  else if (text.length() <= width){
     result = text;
  }
  else {
     String lineBreaker = "\n";
     String wrappedLine = text.substring(0, width) +
                                     lineBreaker;
     String remainingLines = text.substring(width);
     result = wrappedLine + wordWrap(remainingLines, width);
  }
  return result;
}
```

LeanMind
EFFICIENT PEOPLE

~~3. Variables definidas siempre arriba!~~

```
public String wordWrap(String text, int width){
   String wrappedLine;
   String remainingLines;

   if (text == null){
      return "";
   }
   if (text.length() <= width){
      return text;
   }
   wrappedLine = text.substring(0, width) + "\n";
   remainingLines = text.substring(width);
   return wrappedLine + wordWrap(remainingLines, width);
}
```

```
public String wordWrap(String text, int width){
   if (text == null){
      return "";
   }
   if (text.length() <= width){
      return text;
   }
   String wrappedLine = text.substring(0, width) + "\n";
   String remainingLines = text.substring(width);
   return wrappedLine + wordWrap(remainingLines, width);
}
```

LeanMind
EFFICIENT PEOPLE

~~4. Asignar siempre un valor por defecto!~~

```
public String wordWrap(String text, int width){
   String wrappedLine = "";
   String remainingLines = "";

   if (text == null){
      return "";
   }
   if (text.length() <= width){
      return text;
   }
   wrappedLine = text.substring(0, width) + "\n";
   remainingLines = text.substring(width);
   return wrappedLine + wordWrap(remainingLines, width);
}
```

```
public String wordWrap(String text, int width){
   if (text == null){
      return "";
   }
   if (text.length() <= width){
      return text;
   }
   String wrappedLine = text.substring(0, width) + "\n";
   String remainingLines = text.substring(width);
   return wrappedLine + wordWrap(remainingLines, width);
}
```

LeanMind
EFFICIENT PEOPLE

~~4. No "escribir" en los parámetros de las funciones!~~

```
public String parseTemplate(String template,
                            Map<String, String> variables){
  String trimmedTemplate = template.trim();
  String resultTemplate = trimmedTemplate;
  for (Entry<String, String> pair : variables.entrySet()){
     resultTemplate = resultTemplate.replace(
                                  pair.getKey(), pair.getValue());
  }
  return resultTemplate;
}
```

```
public String parseTemplate(String template,
                            Map<String, String> variables){
  template = template.trim();
  for (Entry<String, String> pair : variables.entrySet()){
     template  = template.replace(
                            pair.getKey(), pair.getValue());
  }
  return template;
}
```

LeanMind
EFFICIENT PEOPLE

~~5. Las interfaces desacoplan~~

```
public interface ILogger {...}
public class Logger : ILogger {...}

public interface Logger {...}
public class LoggerImpl implements Logger {...}
```

```
public interface Logger {...}

public class FileLogger : Logger {...}

public class SqlLogger : Logger {...}
```

~~6. Getters y setters para todo~~

```
public class Address {
   private String streetName;

   public String getStreetName(){
      return streetName;
   }
   public void setStreetName(String value){
      this.streetName = value;
   }
}
```

```
public class Address {
   private String streetName;

   public Address(String streetName){
        this.streetName = streetName;
   }
   public Address changeStreetName(String streetName){
        return new Address(streetName);
   }
   public bool isSameStreet(String streetName){
        return this.streetName.equals(streetName);
   }
}
```

LeanMind
EFFICIENT PEOPLE

~~Maximizar la eficiencia!~~

~~Minimizar los ciclos de CPU!~~

~~Comentarios en el código!~~

~~Explicar todo lo que hace el código!~~

Conclusiones:

- Conocer el **contexto**
- Razonar el **por qué**
- Pensamiento **crítico** sobre dogma
- **Mantenibilidad** sobre eficiencia