E1.1

E1.1

```
err1() {
  return "¿Dónde está el error?";
}
```

JShell:

```
jshell> err1(){
   ...>     return "¿Dónde está el error?";
   ...> }
|  Error:
|  ';' expected
|  err1(){
|        ^
|  Error:
|  cannot find symbol
|    symbol:   method err1()
|  err1(){
|  ^--^

jshell>
```

Solución: Debemos definir el valor de retorno

```
jshell> String err1(){
   ...>     return "¿Dónde está el error?";
   ...> }
|  created method err1()

jshell> err1();
$2 ==> "¿Dónde está el error?"

jshell>
```

E1.2

E1.2

```
string err2() {
  return "¿String o string?";
}
```

JShell:

```
jshell> string err2(){
   ...>     return "¿String o string?";
   ...> }
|  modified method err2(), however, it cannot be referenced until class string is declared

jshell>
```

Solución: String

```
jshell> String err2(){
   ...>     return "¿String o string?";
   ...> }
|  replaced method err2()

jshell> err2();
$7 ==> "¿String o string?"

jshell>
```

E1.3

E1.3

```
String err3(who) {
  return String.format("%s, cuál es el error?", who);
}
```

JShell:

```
jshell> String err3(who) {
   ...>     return String.format("%s, cuál es el error?",who);
   ...> }
|  Error:
|  <identifier> expected
|  String err3(who) {
|                  ^

jshell>
```

Solución: Debemos definir el tipo de valor que recibe como parámetro la función.

```
jshell> String err3(String who) {
   ...>     return String.format("%s, cuál es el error?",who);
   ...> }
|  created method err3(String)

jshell> err3("Código");
$9 ==> "Código, cuál es el error?"

jshell> err3("Luis");
$10 ==> "Luis, cuál es el error?"

jshell>
```

E1.4

E1.4

```
String err4() {
  return "Este es un error sutil"
}
```

JShell:

```
jshell> String err4(){
   ...>     return "Este es un error sutil"
   ...> }
|  Error:
|  ';' expected
|      return "Este es un error sutil"
|                                     ^

jshell>
```

Solución: El famoso punto y coma.

```
jshell> String err4(){
   ...>     return "Este es un error sutil";
   ...> }
|  created method err4()

jshell> err4();
$12 ==> "Este es un error sutil"

jshell>
```

E1.5

E1.5

```
String err5() {
  "Es es un poco menos sutil";
}
```

JShell:

```
jshell> String err5(){
   ...>     "Es un poco menos sutil";
   ...> }
|  Error:
|  not a statement
|      "Es un poco menos sutil";
|      ^--------------------^
|  Error:
|  missing return statement
|  String err5(){
|              ^
```

Solución: cuando es una función de retorno debemos utilizar el return al finalizar el flujo de la función.

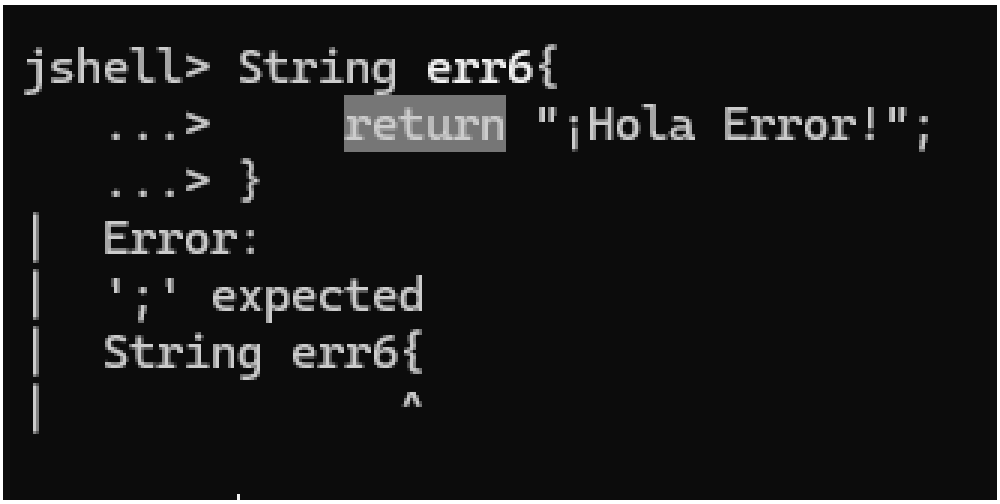
```
jshell> String err5(){
   ...>     return "Es un poco menos sutil";
   ...> }
|  created method err5()

jshell> err5();
$14 ==> "Es un poco menos sutil"

jshell>
```

E1.6

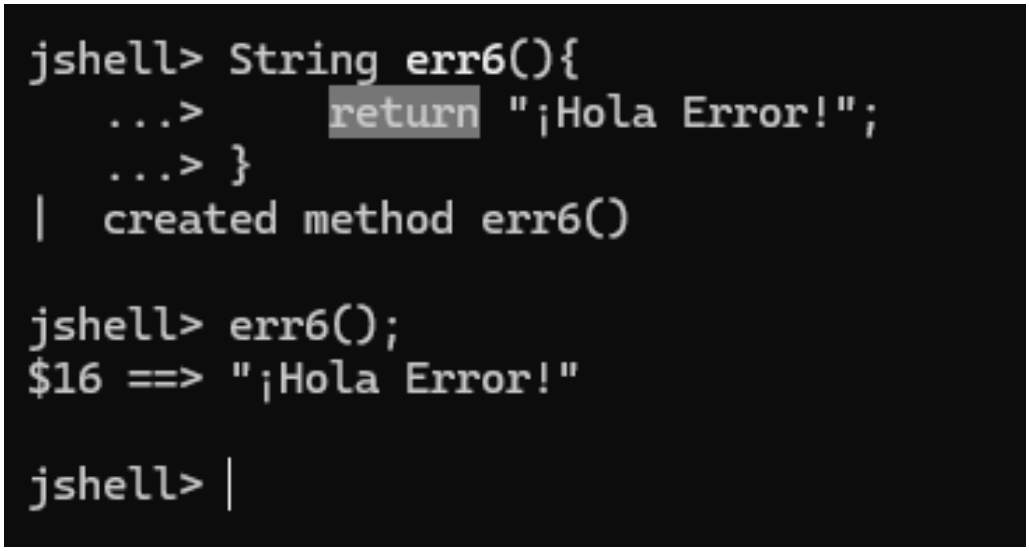
E1.6

```
String err6 {
  return "¡Hola Error!";
}
```

JShell:



Solución: Lista de parámetros siempre debe de ir declarada aun así no reciba parámetros de entrada.



E1.7

E1.7

```
String err7(String quien) {
  return format("%s ¿Un poco mañoso, no?", quien);
}
```

JShell:

```
jshell> String err7(String quien){
   ...>     return format("%s ¿Un poco mañoso, no?",quien);
   ...> }
|  created method err7(String), however, it cannot be invoked until method format(java.lang.String,java.lang.String) is declared

jshell>
```

Solución: format es un método estático de la clase String.

```
jshell> String err7(String quien){
   ...>     return String.format("%s ¿Un poco mañoso, no?",quien);
   ...> }
|  modified method err7(String)

jshell> err7("Luis");
$19 ==> "Luis ¿Un poco mañoso, no?"

jshell>
```

Ejercicios de codificación

E1.7

E1.7

Llena los huecos para crear una función que dados 3 Strings que sean emojis, los devuelva en formato "[%s, %s, %s]" por ejemplo al invocar la función así: adivinaLaPelicula("❄️", "🚢", "💑"), retornaría "[❄️, 🚢, 💑]". Para cada hueco con una letra, escoge el reemplazo utilizando una de las opciones numeradas. Una vez que utilices una opción numerada ya no la puedes usar en otro reemplazo.

```
String _A_(String emoji1_B_ String emoji2_C_ String emoji3) _D_
  return _E_("[%s, %s, %s]"_F_ emoji1_G_ emoji2_H_ emoji3);
_I_
```

1. ,
2. ,
3. ,
4. ,
5. ,
6. } → I
7. { → D
8. adivinaLaPelicula → A
9. String.format → E

JShell solución:

```
jshell> String adivinaLaPalabra(String emoji1,String emoji2,String emoji3){
   ...>     return String.format("[%s,%s,%s]",emoji1,emoji2,emoji3);
   ...> }
|  created method adivinaLaPalabra(String,String,String)

jshell> adivinaLaPalabra("😳","🐂","🐓");
$21 ==> "[😳, 🐂, 🐓]"

jshell>
```

E1.8

Llena los huecos, para crear una función que concatene la cadena "こんにちは" (hola en japonés) y un parámetro dado. Para cada hueco con una letra, escoge el reemplazo utilizando una de las opciones numeradas. Una vez que utilices una opción numerada ya no la puedes usar en otro reemplazo.

23

```
_A_ _B_ {
_C_ String.format("こんにちは_D_", _E_)_F_
    }
```

1. return
2. ;
3. dare
4. konnichiwa(String dare)
5. %s
6. String

Solución:

```
_A_ _B_ {
_C_ String.format("こんにちは_D_", _E_)_F_
    }
```

1. return =C
2. ; =F
3. dare =E
4. konnichiwa(String dare) =B
5. %s =D
6. String =A

JShell:

```
jshell> String konnichiwa(String dare){
   ...>     return String.format("こんにちは %s",dare);
   ...> }
|  created method konnichiwa(String)

jshell> konnichiwa("Luis Carlos");
$23 ==> "こんにちは Luis Carlos"

jshell>
```

E1.9

E1.9
Escribe una función separador() que retorne el String "<<<>>>"

`> separador()`	`"<<<>>>"`

Solución JShell:

```
jshell> String separador(){
   ...>     return "<<<>>>";
   ...> }
|  created method separador()

jshell> separador();
$25 ==> "<<<>>>"

jshell>
```

E1.10

E1.10
Escribe una función vocales() que retorne el String "aeiou"

`> vocales()`	`"aeiou"`

Solución JShell:

```
jshell> String vocales(){
   ...>     return "aeiou";
   ...> }
|  created method vocales()

jshell> vocales();
$27 ==> "aeiou"

jshell>
```

E1.11

E1.11
Escribe una función tituloAdornado() que tome un String como parámetro y retorne el mismo String precedido por "<<< " y antecedido por " >>>"

`> tituloAdornado("Java")`	`"<<< Java >>>"`
`> tituloAdornado(">>> <<<")`	`"<<< >>> <<< >>>"`
`> tituloAdornado("A + B")`	`"<<< A + B >>>"`

Solución JShell:

```
jshell> String tituloAdornado(String valor){
   ...>     return String.format("<<< %s >>>",valor);
   ...> }
|  modified method tituloAdornado(String)

jshell> tituloAdornado("Java");
$42 ==> "<<< Java >>>"

jshell> tituloAdornado(">>> <<<");
$43 ==> "<<< >>> <<< >>>"

jshell> tituloAdornado("A + B");
$44 ==> "<<< A + B >>>"

jshell>
```

Ejercicios de Corrección de Código

E1.12

E1.12

> adornar("Java")	"<=Java=>"
> adornar("Mundo")	"<=Mundo=>"

```
String adornar(String frase) {
  return String.format("<=%s=>, frase");
}
```

```
jshell> String adornar(String frase){
   ...>     return String.format("<=%s=>,frase");
   ...> }
|  created method adornar(String)

jshell> adornar("Java");
|  Exception java.util.MissingFormatArgumentException: Format specifier '%s'
|        at Formatter.format (Formatter.java:2760)
|        at Formatter.format (Formatter.java:2698)
|        at String.format (String.java:4468)
|        at adornar (#32:2)
|        at (#33:1)

jshell>
```

Solución: El llamado de los parámetros de entrada no van en comillas.

```
jshell> String adornar(String frase){
   ...>     return String.format("<=%s=>",frase);
   ...> }
|  modified method adornar(String)

jshell> adornar("Java");
$35 ==> "<=Java=>"

jshell>
```

E1.13

E1.13

> concatenar("Hola","Mundo")	"[Hola+Mundo]"
> concatenar("A", "Z")	"[A+Z]"

```
String concatenar(String s1, String s2) {
  return String.format("[%s+%s]", s2, s1);
}
```

```
jshell> String concatenar(String s1,String s2){
   ...>     return String.format("[%s+%s]",s2,s1);
   ...> }
|  created method concatenar(String,String)

jshell> concatenar("Hola","Mundo");
$37 ==> "[Mundo+Hola]"

jshell>
```

Solución: Solo cambiar el orden del llamado de los parámetros de entrada.

```
jshell> String concatenar(String s1,String s2){
   ...>     return String.format("[%s+%s]",s1,s2);
   ...> }
|  modified method concatenar(String,String)

jshell> concatenar("Hola","Mundo");
$39 ==> "[Hola+Mundo]"

jshell> concatenar("A","Z");
$40 ==> "[A+Z]"

jshell>
```