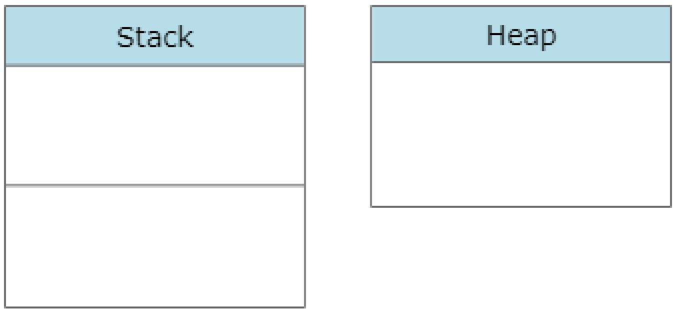
Stage 1:

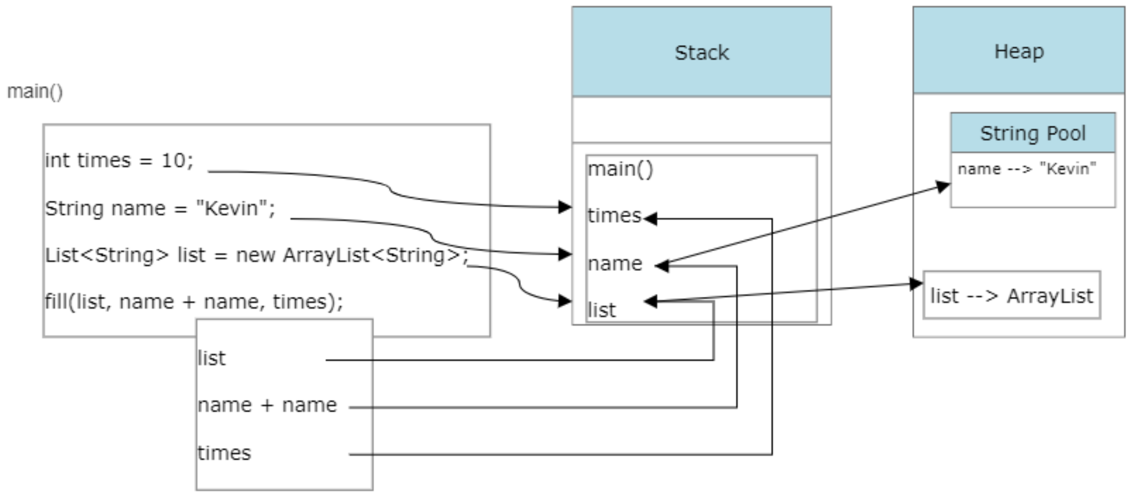
Java Virtual Machine (JVM) divides the memory into different categories like Stack, Heap, Register, Class, Native method areas to manage memory optimally.



Stage 2:

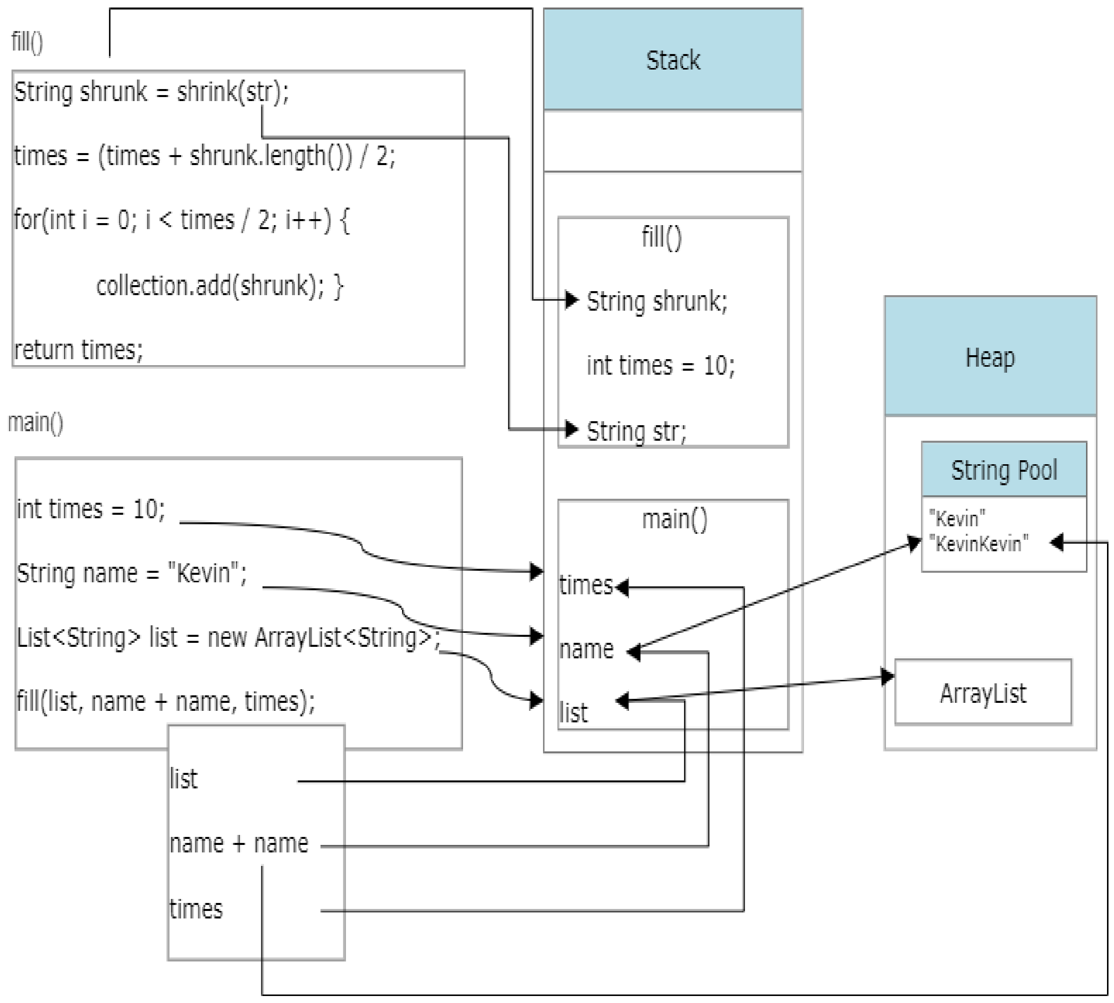
In Stack area, memory will be allocated for all primitives and references that are part of main() before its execution.

- Primitive type variable "times" of int type will be allocated with memory in the stack directly.
- Reference variable "name" of the String type is created in Stack but its value is stored in "String Pool" which is part of "Heap" area.
- Reference variable "list" of the String type is also created in Stack but points to object located on Heap.
- fill(list, name + name, times), is invoked from main(), which passes a reference from the stack of the main() to the ArrayList object with generic type String, references to the name and list object from the String Pool, and copies the primitive type times.



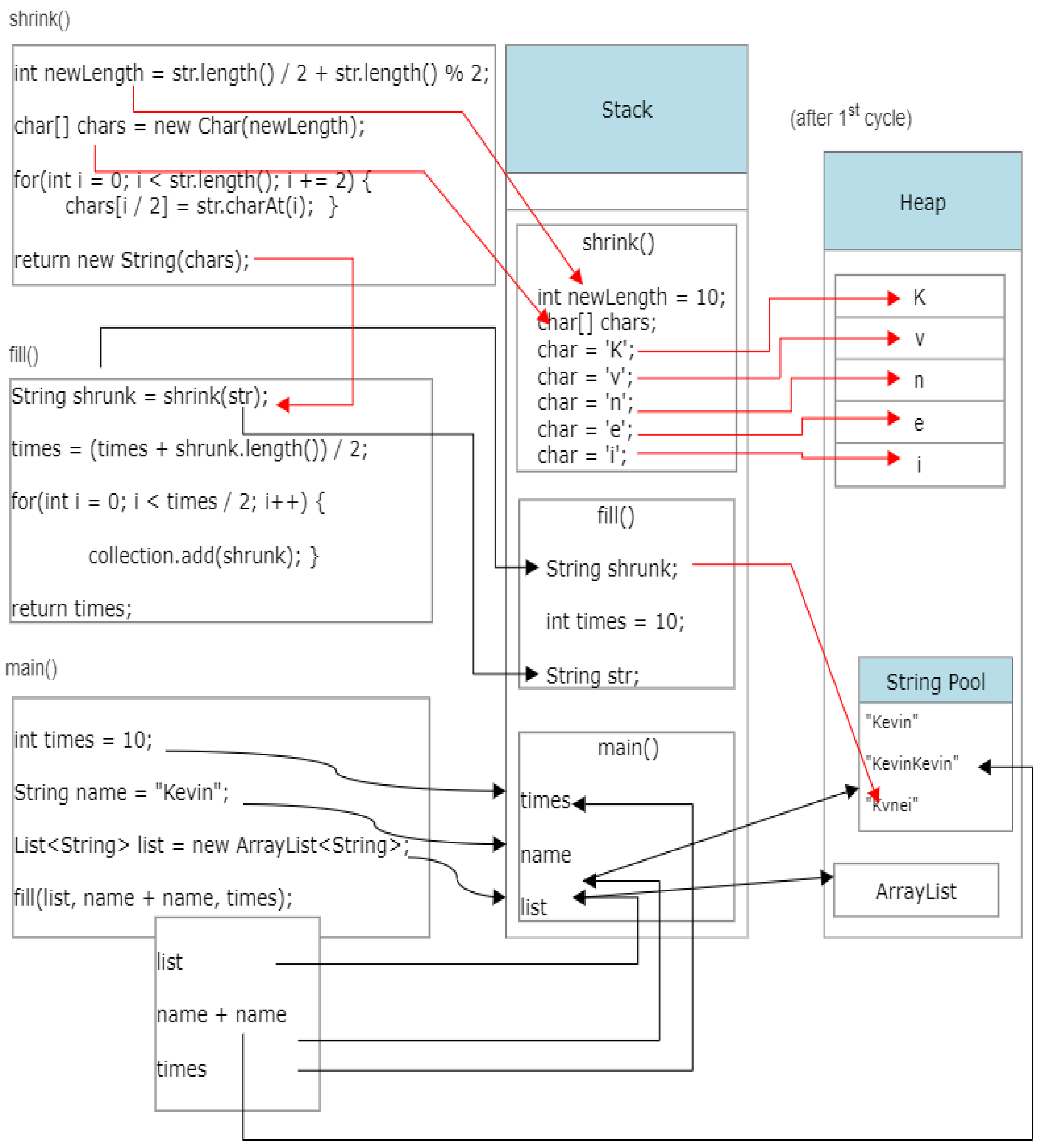
Stage 3:

Upon invoking the fill(), a new "block" of memory is allocated in the stack on top of main(), following LIFO principle of Stack.


fill()
String shrunk = shrink(str);
times = (times + shrunk.length()) / 2;
for(int i = 0; i < times / 2; i++) {
collection.add(shrunk); }
return times;
main()
int times = 10;
String name = "Kevin";
List<String> list = new ArrayList<String>;
fill(list, name + name, times);
list
name + name
times
Stack
fill()
String shrunk;
int times = 10;
String str;
main()
times
name
list
Heap
String Pool
"Kevin"
"KevinKevin"
ArrayList

Stage 4:

Upon invoking the shrink(), a new "block" of memory is allocated in the stack on top of fill(), following LIFO principle of Stack.



shrink()

```
int newLength = str.length() / 2 + str.length() % 2;

char[] chars = new Char(newLength);

for(int i = 0; i < str.length(); i += 2) {
        chars[i / 2] = str.charAt(i);  }

return new String(chars);
```

fill()

```
String shrunk = shrink(str);

times = (times + shrunk.length()) / 2;

for(int i = 0; i < times / 2; i++) {

            collection.add(shrunk); }

return times;
```

main()

```
int times = 10;

String name = "Kevin";

List<String> list = new ArrayList<String>;

fill(list, name + name, times);
```

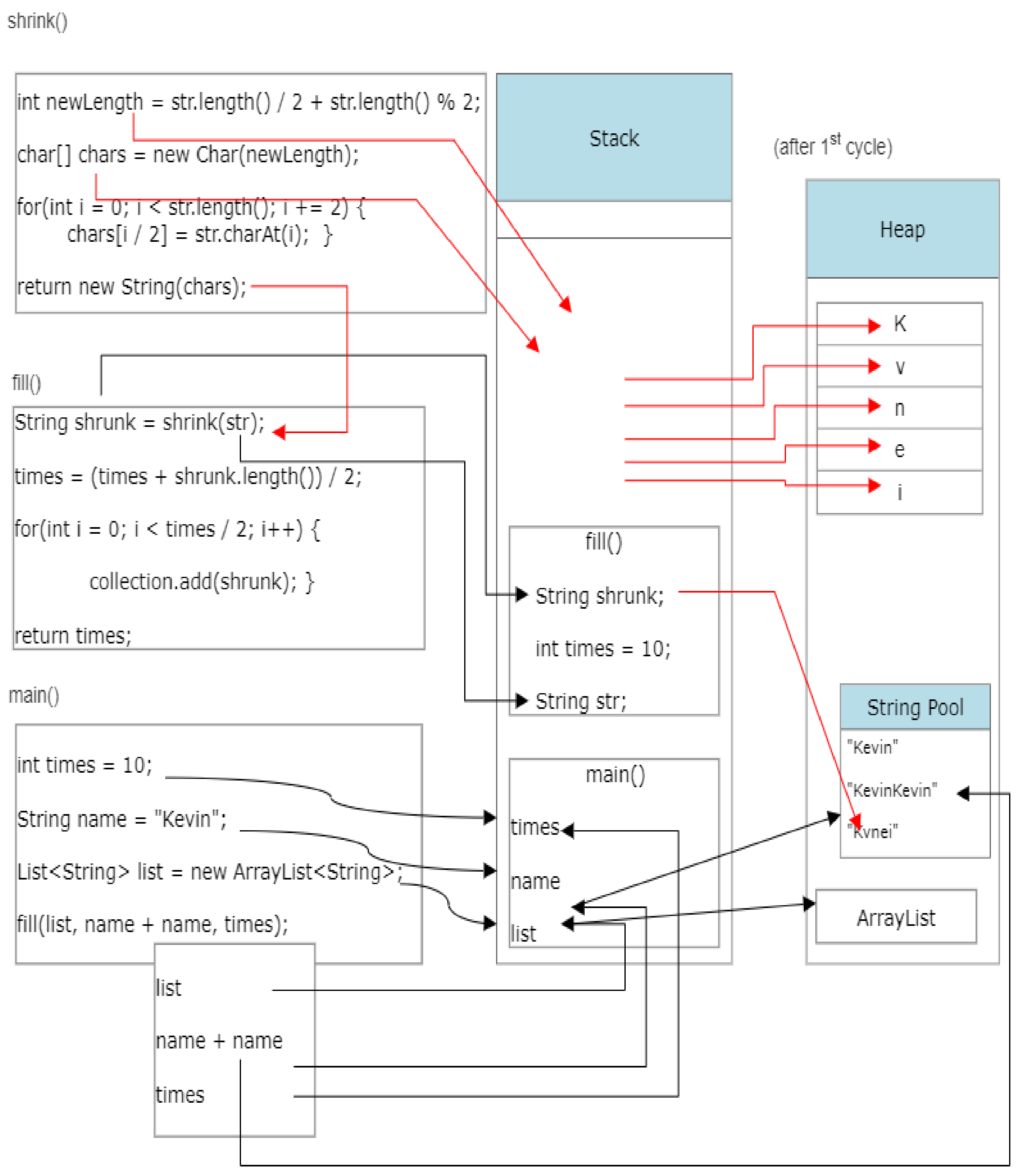
```
list

name + name

times
```

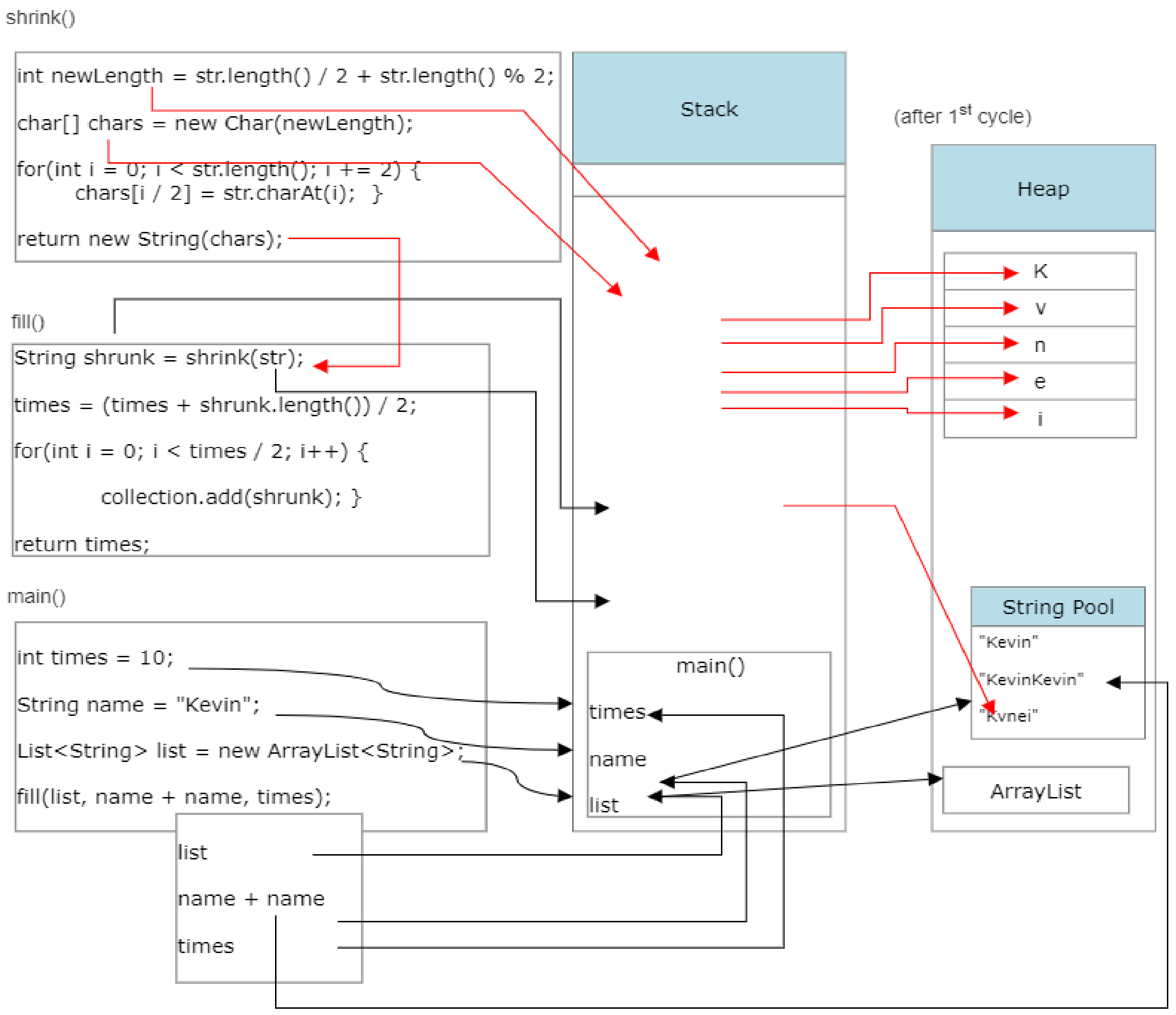
Stage 5:

After the shrink() complete its execution, JVM frees up the memory area associated with it.


shrink()
int newLength = str.length() / 2 + str.length() % 2;
char[] chars = new Char(newLength);
for(int i = 0; i < str.length(); i += 2) {
chars[i / 2] = str.charAt(i); }
return new String(chars);
fill()
String shrunk = shrink(str);
times = (times + shrunk.length()) / 2;
for(int i = 0; i < times / 2; i++) {
collection.add(shrunk); }
return times;
main()
int times = 10;
String name = "Kevin";
List<String> list = new ArrayList<String>;
fill(list, name + name, times);
list
name + name
times
Stack
fill()
String shrunk;
int times = 10;
String str;
main()
times
name
list
(after 1st cycle)
Heap
K
v
n
e
i
String Pool
"Kevin"
"KevinKevin"
"Kvnei"
ArrayList

Stage 6:

After the fill() complete its execution, JVM frees up the memory area associated with it.



shrink()
int newLength = str.length() / 2 + str.length() % 2;
char[] chars = new Char(newLength);
for(int i = 0; i < str.length(); i += 2) {
chars[i / 2] = str.charAt(i); }
return new String(chars);
fill()
String shrunk = shrink(str);
times = (times + shrunk.length()) / 2;
for(int i = 0; i < times / 2; i++) {
collection.add(shrunk); }
return times;
main()
int times = 10;
String name = "Kevin";
List<String> list = new ArrayList<String>;
fill(list, name + name, times);
list
name + name
times
Stack
main()
times
name
list
(after 1st cycle)
Heap
K
v
n
e
i
String Pool
"Kevin"
"KevinKevin"
"Kvnei"
ArrayList

Stage 7:

After the main() complete its execution, JVM frees up the memory area associated with it. Program will be terminated and Stack & Heap memory areas will be cleared.