WORKED EXAMPLE 5.1 Extracting the Middle

Problem Statement Your task is to extract a string containing the middle character from a given string `str`. For example, if the string is `"crate"`, the result is the string `"a"`. However, if the string has an even number of letters, extract the middle two characters. If the string is `"crates"`, the result is `"at"`.

Step 1 Decide on the branching condition.

We need to take different actions for strings of odd and even length. Therefore, the condition is

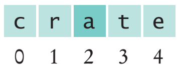
```
Is the length of the string odd?
```

In Java, you use the remainder of division by 2 to find out whether a value is even or odd. Then the test becomes

```
str.length() % 2 == 1?
```

Step 2 Give pseudocode for the work that needs to be done when the condition is true.

We need to find the position of the middle character. If the length is 5, the position is 2.



In general,

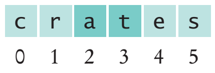
```
position = str.length() / 2 (with the remainder discarded)
result = str.substring(position, position + 1)
```

Step 3 Give pseudocode for the work (if any) that needs to be done when the condition is *not* true.

Again, we need to find the position of the middle character. If the length is 6, the starting position is 2, and the ending position is 3. That is, we would call

```
result = str.substring(2, 4);
```

(Recall that the second parameter of the `substring` method is the first position that we do not extract.)



In general,

```
position = str.length() / 2 - 1
result = str.substring(position, position + 2)
```

Step 4 Double-check relational operators.

Do we really want `str.length() % 2 == 1`? For example, when the length is 5, `5 % 2` is the remainder of the division 5 / 2, which is 1. In general, dividing an odd number by 2 leaves a remainder of 1. (Actually, dividing a negative odd number by 2 leaves a remainder of –1, but the string length is never negative.) Therefore, our condition is correct.

Step 5 Remove duplication.

Here is the statement that we have developed:

```
If str.length() % 2 == 1
    position = str.length() / 2 (with remainder discarded)
    result = str.substring(position, position + 1)
Else
    position = str.length() / 2 - 1
    result = str.substring(position, position + 2)
```

The second statement in each branch is almost identical, but the length of the substring differs. Let's set the length in each branch:

```
If str.length() % 2 == 1
    position = str.length() / 2 (with remainder discarded)
    length = 1
Else
    position = str.length() / 2 - 1
    length = 2
result = str.substring(position, position + length)
```

Step 6 Test both branches.

We will use a different set of strings for testing. For an odd-length string, consider `"monitor"`. We get

```
position = str.length() / 2 = 7 / 2 = 3 (with remainder discarded)
length = 1
result = str.substring(3, 4) = "i"
```

For the even-length string `"monitors"`, we get

```
position = str.length() / 2 - 1 = 8 / 2 - 1 = 3 (with remainder discarded)
length = 2
result = str.substring(3, 5) = "it"
```

Step 7 Assemble the `if` statement in Java.

Here's the completed code segment:

```
if (str.length() % 2 == 1)
{
   position = str.length() / 2;
   length = 1;
}
else
{
   position = str.length() / 2 - 1;
   length = 2;
}
String result = str.substring(position, position + length);
```

You can find the complete program in the `ch05/worked_example_1` directory of the book's companion code.