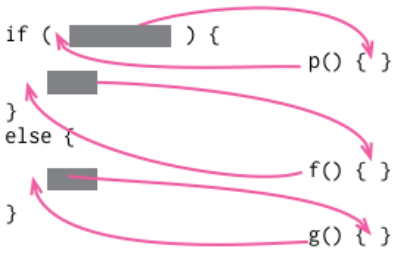
Decompose Conditional

previous:
Simplifying Conditional Logic

next:
Consolidate Conditional Expression




```
if (!aDate.isBefore(plan.summerStart) && !aDate.isAfter(plan.summerEnd))
  charge = quantity * plan.summerRate;
else
  charge = quantity * plan.regularRate + plan.regularServiceCharge;
```

```
if (summer())
  charge = summerCharge();
else
  charge = regularCharge();
```

Motivation

One of the most common sources of complexity in a program is complex conditional logic. As I write code to do various things depending on various conditions, I can quickly end up with a pretty long function. Length of a function is in itself a factor that makes it harder to read, but conditions increase the difficulty. The problem usually lies in the fact that the code, both in the condition checks and in the actions, tells me what happens but can easily obscure *why* it happens.

As with any large block of code, I can make my intention clearer by decomposing it and

replacing each chunk of code with a function call named after the intention of that chunk. With conditions, I particularly like doing this for the conditional part and each of the alternatives. This way, I highlight the condition and make it clear what I'm branching on. I also highlight the reason for the branching.

This is really just a particular case of applying Extract Function to my code, but I like to highlight this case as one where I've often found a remarkably good value for the exercise.

Mechanics

- Apply Extract Function on the condition and each leg of the conditional.

Example

Suppose I'm calculating the charge for something that has separate rates for winter and summer:

```
if (!aDate.isBefore(plan.summerStart) && !aDate.isAfter(plan.summerEnd))
  charge = quantity * plan.summerRate;
else
  charge = quantity * plan.regularRate + plan.regularServiceCharge;
```

I extract the condition into its own function.

```
if (summer())
  charge = quantity * plan.summerRate;
else
  charge = quantity * plan.regularRate + plan.regularServiceCharge;

function summer() {
  return !aDate.isBefore(plan.summerStart) && !aDate.isAfter(plan.summerEnd);
}
```

Then I do the `then` leg:

```
if (summer())
  charge = summerCharge();
else
  charge = quantity * plan.regularRate + plan.regularServiceCharge;

function summer() {
  return !aDate.isBefore(plan.summerStart) && !aDate.isAfter(plan.summerEnd);
}
function summerCharge() {
  return quantity * plan.summerRate;
}
```

Finally, the `else` leg:

```
if (summer())
  charge = summerCharge();
else
  charge = regularCharge();
```

```
function summer() {
  return !aDate.isBefore(plan.summerStart) && !aDate.isAfter(plan.summerEnd);
}
function summerCharge() {
  return quantity * plan.summerRate;
}
function regularCharge() {
  return quantity * plan.regularRate + plan.regularServiceCharge;
}
```

With that done, I like to reformat the conditional using the ternary operator.

```
charge = summer() ? summerCharge() : regularCharge();

function summer() {
  return !aDate.isBefore(plan.summerStart) && !aDate.isAfter(plan.summerEnd);
}
function summerCharge() {
  return quantity * plan.summerRate;
}
function regularCharge() {
  return quantity * plan.regularRate + plan.regularServiceCharge;
}
```

previous:
Simplifying Conditional Logic

next:
Consolidate Conditional Expression