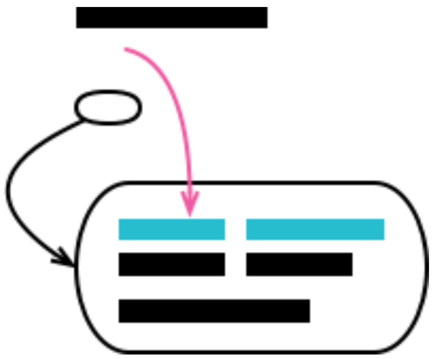
Move Statements into Function

previous:
Move Field

next:
Move Statements to Callers



```
result.push(`<p>title: ${person.photo.title}</p>`);
result.concat(photoData(person.photo));

function photoData(aPhoto) {
  return [
    `<p>location: ${aPhoto.location}</p>`,
    `<p>date: ${aPhoto.date.toDateString()}</p>`,
  ];
}
```

⇓

```
result.concat(photoData(person.photo));

function photoData(aPhoto) {
  return [
    `<p>title: ${aPhoto.title}</p>`,
    `<p>location: ${aPhoto.location}</p>`,
    `<p>date: ${aPhoto.date.toDateString()}</p>`,
  ];
}
```

inverse of: Move Statements to Callers

Motivation

Removing duplication is one of the best rules of thumb of healthy code. If I see the same code executed every time I call a particular function, I look to combine that repeating code into the function itself. That way, any future modifications to the repeating code can be done in one place and used by all the callers. Should the code vary in the future, I can easily move it (or some of it) out again with Move Statements to Callers.

I move statements into a function when I can best understand these statements as part of the called function. If they don't make sense as part of the called function, but still should be called with it, I'll simply use Extract Function on the statements and the called function. That's essentially the same process as I describe below, but without the inline and rename steps. It's not unusual to do that and then, after later reflection, carry out those final steps.

Mechanics

- If the repetitive code isn't adjacent to the call of the target function, use Slide Statements to get it adjacent.
- If the target function is only called by the source function, just cut the code from the source, paste it into the target, test, and ignore the rest of these mechanics.
- If you have more callers, use Extract Function on one of the call sites to extract both the call to the target function and the statements you wish to move into it. Give it a name that's transient, but easy to grep.
- Convert every other call to use the new function. Test after each conversion.
- When all the original calls use the new function, use Inline Function to inline the original function completely into the new function, removing the original function.
- Rename Function to change the name of the new function to the same name as the original function.

  Or to a better name, if there is one.

Example

I'll start with this code to emit HTML for data about a photo:

```
function renderPerson(outStream, person) {
  const result = [];
  result.push(`<p>${person.name}</p>`);
  result.push(renderPhoto(person.photo));
  result.push(`<p>title: ${person.photo.title}</p>`);
  result.push(emitPhotoData(person.photo));
  return result.join("\n");
}
```

```
function photoDiv(p) {
  return [
    "<div>",
    `<p>title: ${p.title}</p>`,
    emitPhotoData(p),
    "</div>",
  ].join("\n");
}

function emitPhotoData(aPhoto) {
  const result = [];
  result.push(`<p>location: ${aPhoto.location}</p>`);
  result.push(`<p>date: ${aPhoto.date.toDateString()}</p>`);
  return result.join("\n");
}
```

This code shows two calls to `emitPhotoData`, each preceded by a line of code that is semantically equivalent. I'd like to remove this duplication by moving the title printing into `emitPhotoData`. If I had just the one caller, I would just cut and paste the code, but the more callers I have, the more I'm inclined to use a safer procedure.

I begin by using Extract Function on one of the callers. I'm extracting the statements I want to move into `emitPhotoData`, together with the call to `emitPhotoData` itself.

```
function photoDiv(p) {
  return [
    "<div>",
    zznew(p),
    "</div>",
  ].join("\n");
}

function zznew(p) {
  return [
    `<p>title: ${p.title}</p>`,
    emitPhotoData(p),
  ].join("\n");
}
```

I can now look at the other callers of `emitPhotoData` and, one by one, replace the calls and the preceding statements with calls to the new function.

```
function renderPerson(outStream, person) {
  const result = [];
  result.push(`<p>${person.name}</p>`);
  result.push(renderPhoto(person.photo));
  result.push(zznew(person.photo));
  return result.join("\n");
}
```

Now that I've done all the callers, I use Inline Function on `emitPhotoData`:

```
function zznew(p) {
  return [
    `<p>title: ${p.title}</p>`,
    `<p>location: ${p.location}</p>`,
    `<p>date: ${p.date.toDateString()}</p>`,
  ].join("\n");
}
```

and finish with Rename Function:

```
  function renderPerson(outStream, person) {
    const result = [];
    result.push(`<p>${person.name}</p>`);
    result.push(renderPhoto(person.photo));
    result.push(emitPhotoData(person.photo));
    return result.join("\n");
  }

  function photoDiv(aPhoto) {
    return [
      "<div>",
      emitPhotoData(aPhoto),
      "</div>",
    ].join("\n");
  }

  function emitPhotoData(aPhoto) {
    return [
      `<p>title: ${aPhoto.title}</p>`,
      `<p>location: ${aPhoto.location}</p>`,
      `<p>date: ${aPhoto.date.toDateString()}</p>`,
    ].join("\n");
  }
```

I also make the parameter names fit my convention while I'm at it.

previous:
Move Field

next:
Move Statements to Callers