Rule: no-shadowed-variable

Disallows shadowing variable declarations.

Rationale

When a variable in a local scope and a variable in the containing scope have the same name, shadowing occurs. Shadowing makes it impossible to access the variable in the containing scope and obscures to what value an identifier actually refers. Compare the following snippets:

```
const a = 'no shadow';
function print() {
    console.log(a);
}
print(); // logs 'no shadow'.
```

```
const a = 'no shadow';
function print() {
    const a = 'shadow'; // TSLint will complain here.
    console.log(a);
}
print(); // logs 'shadow'.
```

ESLint has an equivalent rule. For more background information, refer to this MDN closure doc.

Config

You can optionally pass an object to disable checking for certain kinds of declarations. Possible keys are `"class"`, `"enum"`, `"function"`, `"import"`, `"interface"`, `"namespace"`, `"typeAlias"` and `"typeParameter"`. You can also pass `"underscore"` to ignore variable names that begin with `_`. Just set the value to `false` for the check you want to disable. All checks default to `true`, i.e. are enabled by default. Note that you cannot disable variables and parameters.

The option `"temporalDeadZone"` defaults to `true` which shows errors when shadowing block scoped declarations in their temporal dead zone. When set to `false` parameters, classes, enums and variables declared with `let` or `const` are not considered shadowed if the shadowing occurs within their temporal dead zone.

The following example shows how the `"temporalDeadZone"` option changes the linting result:

```
function fn(value) {
    if (value) {
        const tmp = value; // no error on this line if "temporalDeadZone" is false
        return tmp;
    }
    let tmp = undefined;
    if (!value) {
        const tmp = value; // this line always contains an error
        return tmp;
    }
}
```

Config examples

```
"no-shadowed-variable": true
```

```
"no-shadowed-variable": [
  true,
```

```
  {
    "class": true,
    "enum": true,
    "function": true,
    "interface": false,
    "namespace": true,
    "typeAlias": false,
    "typeParameter": false,
    "underscore": false
  }
]
```

Schema

```
{
  "type": "object",
  "properties": {
    "class": {
      "type": "boolean"
    },
    "enum": {
      "type": "boolean"
    },
    "function": {
      "type": "boolean"
    },
    "import": {
      "type": "boolean"
    },
    "interface": {
      "type": "boolean"
    },
    "namespace": {
      "type": "boolean"
```

```
    },
    "typeAlias": {
      "type": "boolean"
    },
    "typeParameter": {
      "type": "boolean"
    },
    "temporalDeadZone": {
      "type": "boolean"
    },
    "underscore": {
      "type": "boolean"
    }
  }
}
```