Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Ciudad de México

Ingeniería en Tecnologías computacionales
Diseño y Arquitectura de Software

Cheat Sheet

Autores:
Arturo Efrén Jiménez Garibaldy
Ignacio Alvarado Reyes
José A. Berrón Gutiérrez
Aldo Ponce de la Cruz
Manuel Herrasti González

Profesor:
Marlene Ofelia Sánchez Escobar

Extract Method:

Error

```
printOwing(): void {
  printBanner();

  // Print details.
  console.log("name: " + name);
  console.log("amount: " + getOutstanding());
}
```

Solución

```
printOwing(): void {
  printBanner();
  printDetails(getOutstanding());
}

printDetails(outstanding: number): void {
  console.log("name: " + name);
  console.log("amount: " + outstanding);
}
```

Motivación:
Entre más líneas tenga un método, más difícil de entender. Es por eso que es mejor extraer funciones del método y crear nuevos métodos a partir de ahí. De esta manera el método es más claro a la hora de leerlo.

Mecánica:

1. Creas un nuevo método con un nombre bastante descriptivo de su funcionalidad.
2. Extraes la función del método previo al nuevo método, borras el código del método anterior y haces una llamada al nuevo método.
3. En caso de que cambies alguna variable en el nuevo método, asegúrate de devolver esa variable para que todo lo demás funcione como debe.

Replace Temp with Query:

Error

```
calculateTotal(): number {
  let basePrice = quantity * itemPrice;
  if (basePrice > 1000) {
    return basePrice * 0.95;
  }
  else {
    return basePrice * 0.98;
  }
}
```

Solución

```
calculateTotal(): number {
  if (basePrice() > 1000) {
    return basePrice() * 0.95;
  }
  else {
    return basePrice() * 0.98;
  }
}
basePrice(): number {
  return quantity * itemPrice;
}
```

Motivación:
Va bastante de la mano con el método **Extract Method**. Similar a **Extract Method**, mueves la funcionalidad del cálculo de algo a un método, de esta manera los otros métodos que lleguen a utilizar esta funcionalidad hacen una “query” al dato.

Mecánica:

1. Primero asegurarse que solo una vez ocurre una asignación a la variable dentro del método.
2. Usamos **Extract Method** para poner la función en un nuevo método. Tenemos que asegurarnos que el nuevo método solo devuelve un valor y no modifica nada.
3. Reemplazamos la variable con la query que hacemos al nuevo método.

Extract Variable:

Error

```
renderBanner(): void {
  if ((platform.toUpperCase().indexOf("MAC") > -1) &&
       (browser.toUpperCase().indexOf("IE") > -1) &&
        wasInitialized() && resize > 0 )
  {
    // do something
  }
}
```

Solución

```
renderBanner(): void {
  const isMacOs = platform.toUpperCase().indexOf("MAC") > -1;
  const isIE = browser.toUpperCase().indexOf("IE") > -1;
  const wasResized = resize > 0;

  if (isMacOs && isIE && wasInitialized() && wasResized) {
    // do something
  }
}
```

Motivación:
Se hace con el objetivo de hacer una expresión compleja más simple al dividir sus partes intermedias.

Mecánica:
1. Se ingresa una nueva línea antes de la expresión y se declara una nueva variable en esa línea. A esa variable se le asigna parte de la complejidad de la expresión.
2. En la expresión se reemplaza esa parte de su expresión con la nueva variable.
3. Se repite el proceso para todas las partes de la expresión.

Split Temporary Variable

Error

```
let temp = 2 * (height + width);
console.log(temp);
temp = height * width;
console.log(temp);
```

Solución

```
const perimeter = 2 * (height + width);
console.log(perimeter);
const area = height * width;
console.log(area);
```

Motivación:
Se implementa esta técnica de refactorización porque al tener diferentes variables para cada una de las variaciones de la misma, es más fácil encontrar el error en lugar de seguir todo el camino que una variable pasó por los cambios. Si es que se necesita algún cambio se tiene que seguir desde la primera definición de la variable y sus iteraciones y cambios de valores. Cada componente del programa es responsable de una cosa solamente y así se puede manipular más fácilmente el código en caso de pruebas.

Mecánica:

1. Primero es encontrar la variable con un valor, si encontramos esa misma variable pero definida de otra manera, procedemos a renombrarla para que corresponda con lo que se asigna
2. Cambiamos los campos en donde el otro era utilizado y actualizamos con el nuevo nombre
3. Se repite este procedimiento para cada una de las variables que cumplan con las definiciones

Remove Assignments to Parameters

Error

```
int discount(int inputVal, int quantity)
  if (quantity > 50) {
    inputVal -= 2;
  }
  // ...
}
```

Solución

```
int discount(int inputVal, int quantity)
  int result = inputVal;
  if (quantity > 50) {
    result -= 2;
  }
  // ...
}
```

Motivación:
Esta técnica de refactorización se utiliza para evitar cambiar la referencia de un objeto que ha sido pasado como argumento en un método, así como su valor. Es por esto que la asignación directa a una variable que ha sido recibida como argumento debe evitarse, ya que podría llevar a problemas difíciles de trackear debido a un cambio inesperado de valor.

Mecánica:
1. Se crea una variable temporal para el parámetro o variable que ha sido pasado como argumento del método.
2. Se reemplaza todas las referencias al parámetro por la variable temporal recientemente creada.
3. Se realiza la asignación del parámetro del método a la variable temporal.

Replace Method with Method Object:

Error

```
renderBanner(): void {
  if ((platform.toUpperCase().indexOf("MAC") > -1) &&
       (browser.toUpperCase().indexOf("IE") > -1) &&
        wasInitialized() && resize > 0 )
  {
    // do something
  }
}
```

Solución

```
renderBanner(): void {
  const isMacOs = platform.toUpperCase().indexOf("MAC") > -1;
  const isIE = browser.toUpperCase().indexOf("IE") > -1;
  const wasResized = resize > 0;

  if (isMacOs && isIE && wasInitialized() && wasResized) {
    // do something
  }
}
```

Motivación:
Se tiene un método muy largo el cual sus variables están demasiado entrelazadas que no se puede aplicar **Extract Method**.

Mecánica:
1. Se crea una clase. Asegurarse que el nombre de la clase tenga que ver con el propósito de la refactorización.

2. En la nueva clase se crea un campo privado para guardar una referencia de la instancia de la clase previa.
3. Se crean campos privados separados para cada variable local del método.
4. Se crea un constructor que acepte como parámetro los valores de todas las variables locales del método y que también inicialice las variables privadas.
5. Se declara el método principal y se copia el código del método original, reemplazando las variables con las variables privadas.
6. Se reemplaza el cuerpo del método original en la clase original con el método del nuevo objeto creado.

Replace Magic Number with Symbolic Constant:

Error

```
potentialEnergy(mass: number, height: number): number {
  return mass * height * 9.81;
}
```

Solución

```
static const GRAVITATIONAL_CONSTANT = 9.81;

potentialEnergy(mass: number, height: number): number {
  return mass * height * GRAVITATIONAL_CONSTANT;
}
```

Motivación:
Se entiende por “Magic Number” a un número que se encuentra en el código pero que no se sabe su significado, haciendo difícil de entender el por qué se usa. Luego el problema puede ser si de debe de cambiar este número, pues puede estar presente en varias partes del código y se tendrían que cambiar todas las instancias en donde se usa ese número mágico

Mecánica:
1. Declara una constante y asígnale el valor del número mágico
2. Encuentra todos los usos del número mágico.
3. Si el número mágico representa el uso y valor de la constante, reemplaza el número por la constante.