Chapter 5 – Type Narrowing

1. Introduction
   1. Type narrowing is when TypeScript can infer more specific types based on the variable’s surrounding code.

|  |
| --- |
| function formatDate(date: string | number) {  // date can be a number or string here    if (typeof date === 'string') {  // date must be a string here  }  } |

1. Type Guards
   1. One way that TypeScript can narrow a type is with a conditional statement that checks if a variable is a specific type.
   2. TypeScript can recognize typeof type guards that check for these specific values: 'string', 'number', 'boolean', and 'symbol'.

Exercise

|  |
| --- |
| function formatStatistic(stat: string | number) {  if (typeof stat == 'number') {  return stat.toFixed(2);  } else if (typeof stat == 'string') {  return stat.toUpperCase();  }  }  console.log(formatStatistic('Win'));  console.log(formatStatistic(0.364)); |

1. Using in with Type Guards
   1. sometimes we want to see if a specific method exists on a type instead of a type like 'string'
   2. [The in operator](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/in) checks if a property exists on an object itself or anywhere within its prototype chain. Take a look at this example:

Example

|  |
| --- |
| type Tennis = {  serve: () => void;  }    type Soccer = {  kick: () => void;  }    function play(sport: Tennis | Soccer) {  if ('serve' in sport) {  return sport.serve();  }    if ('kick' in sport) {  return sport.kick();  }  } |

Exercise

|  |
| --- |
| type Cat = {  name: string;  run: () => string;  }  type Fish = {  name: string;  swim: () => string;  }  const siameseCat = {  name: 'Proxie',  run: () => 'pitter pat'  }  const bettaFish = {  name: 'Neptune',  swim: () => 'bubble blub'  }  function move(pet: Cat | Fish) {  if ('run' in pet) {  return pet.run();  }  if ('swim' in pet) {  return pet.swim();  }  }  console.log(move(siameseCat)) |

1. Narrowing with else
   1. It turns out that TypeScript can recognize the else block of an if/else statement as being the opposite type guard check of the if statement’s type guard check

Exercise

|  |
| --- |
| function formatPadding(padding: string | number) {  if (typeof padding === 'string') {  return padding.toLowerCase();  } else {  return `${padding}px`;  }  } |

* + 1. The type guard typeof padding === 'string' tells TypeScript that padding within the if statement’s block must be a string
    2. padding must be a number type within the else block

Exercise

|  |
| --- |
| type Pasta = {  menuName: string;  boil: () => string;  }  type Meat = {  menuName: string;  panFry: () => string;  }  const fettuccine = {  menuName: 'Fettuccine',  boil: () => 'Heat water to 212 degrees',  }  const steak = {  menuName: 'New York Strip Steak',  panFry: () => 'Heat oil to 350 degrees',  }  function prepareEntree(entree: Pasta | Meat) {  if ('boil' in entree) {  return entree.boil();  } else {  return entree.panFry();  }  }  console.log(prepareEntree(fettuccine)); |

1. Narrowing After a Type Guard
   1. TypeScript can also type narrow without an else statement, provided that there’s a return statement within the type guard.

Example

|  |
| --- |
| type Tea = {  steep: () => string;  }    type Coffee = {  pourOver: () => string;  }    function brew(beverage: Coffee | Tea) {  if ('steep' in beverage) {  return beverage.steep();  }    beverage.pourOver();  } |

Exercise

|  |
| --- |
| type Metal = {  magnetize: () => string;  }  type Glass = {  melt: () => string;  }  const iron = {  magnetize: () => 'Electromagnet activated'  }  const bottle = {  melt: () => 'Furnace set to 2,700 degrees'  }  function recycle(trash: Metal | Glass) {  // Add your code below:  if ('magnetize' in trash) {  return trash.magnetize();  }    return trash.melt();  }  console.log(recycle(iron)); |

Quiz

1. Which of the following is the best scenario to use the in operator as a type guard?

Graphical user interface, website

Description automatically generated

1. D