Chapter 4 – Using Objects

1. Number Types
2. Constants: *final*
3. Constants: *static final*
4. Integer Division & Powers and Roots
5. Cast and Round
6. Calling *Static* Methods
7. Concatenation in *Print* Statements
8. Converting between Strings and Numbers
9. Reading Input

Number Types

- `int`:
  - integers, no fractional part: `1`, `-4`, `0`
- `double`:
  - floating-point numbers (double precision): `0.5`, `-3.1`

- A numeric computation overflows if the result falls outside the range for the number type:

```
int n = 1000000;
System.out.println(n * n); // prints -
727379968
```

Constants: `final`

- A `final` variable is a constant
- Once its value has been set, it cannot be changed
- Named constants make programs easier to read and maintain
- Convention: Use all-uppercase names for constants

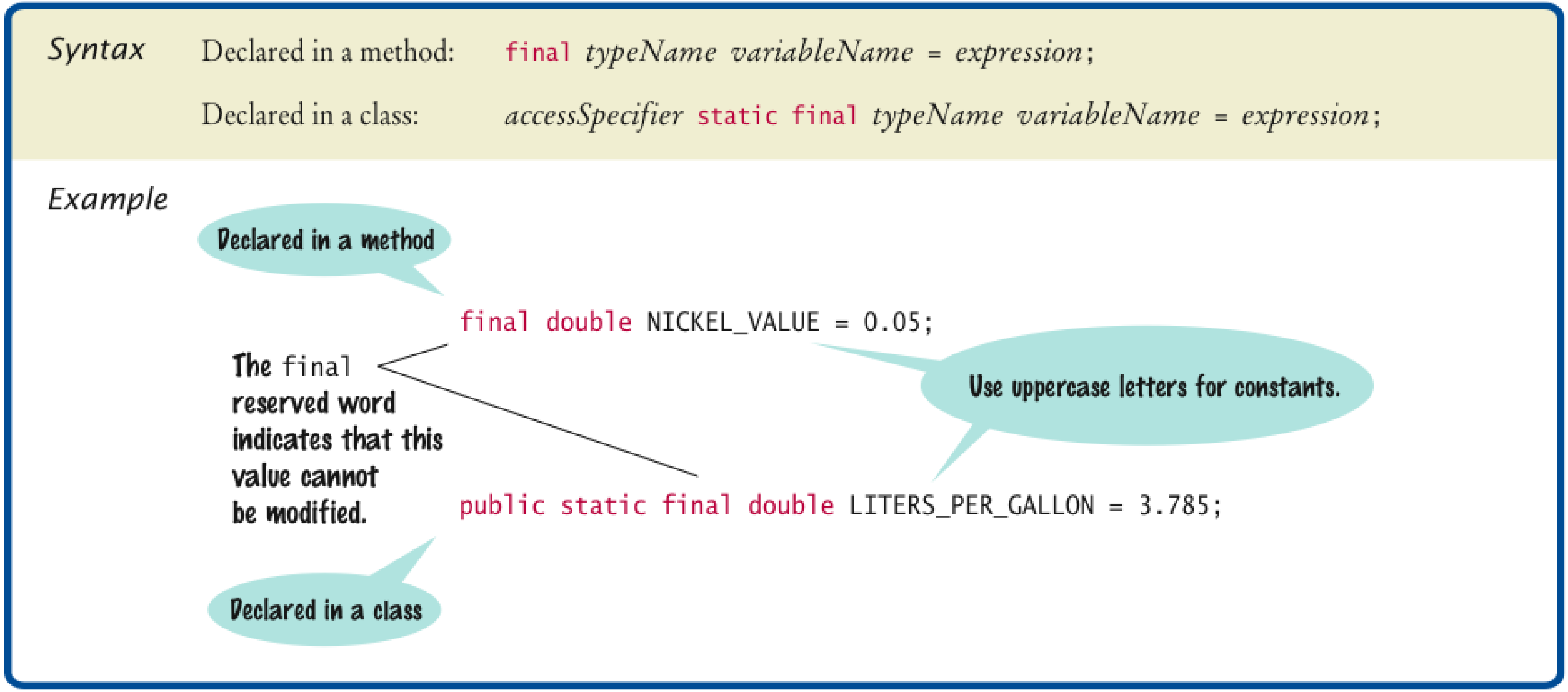
Constants: `static final`

- If constant values are needed in several methods, declare them together with the instance fields of a class and tag them as `static` and `final`
- Give `static final` constants public access to enable other classes to use them

```
  public class Math
{
   . . .
   public static final double E = 2.71828182845904;
   public static final double PI = 3.1415926535897;
}

double circumference = Math.PI * diameter;
```

Constants: `static final`



Syntax Declared in a method: `final` *typeName variableName* = *expression*;

Declared in a class: *accessSpecifier* `static final` *typeName variableName* = *expression*;

Example

```
final double NICKEL_VALUE = 0.05;

public static final double LITERS_PER_GALLON = 3.785;
```

Declared in a method
The final reserved word indicates that this value cannot be modified.
Use uppercase letters for constants.
Declared in a class

Integer Division

- `/` is the division operator
- If both arguments are integers, the result is an integer.

  The remainder is discarded

  `7.0/4` yields `1.75` `7/4` yields `1`
- Get the remainder with `%` (pronounced “modulo”) `7 % 4` is `3`

Powers and Roots

- `Math` class: contains methods `sqrt` and `pow` to compute square roots and powers

```
Math.pow(x, n)

Math.sqrt(x)
```

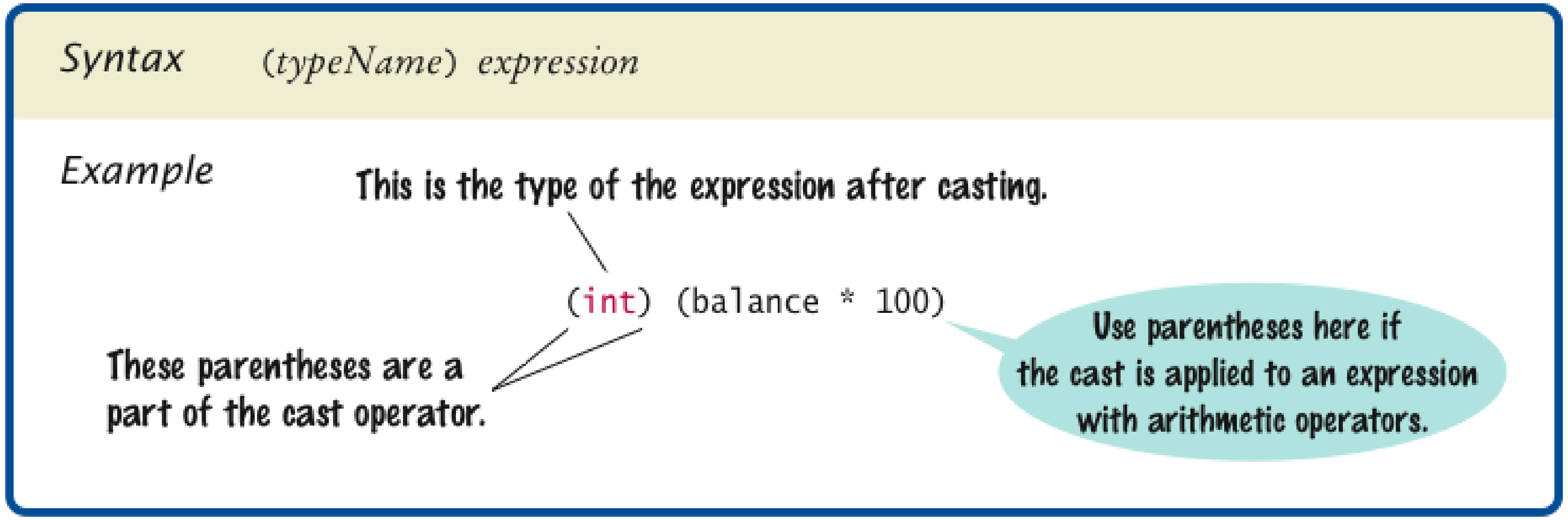
Cast and Round

- <span style="color:red">Cast</span> converts a value to a different type:

```
double balance = total + tax;
int dollars = (int) balance;
```

- `Math.round` converts a floating-point number to nearest integer:

```
long rounded = Math.round(balance);
// if balance is 13.75, then rounded is set to 14
```



Syntax (typeName) expression

Example

```
(int) (balance * 100)
```

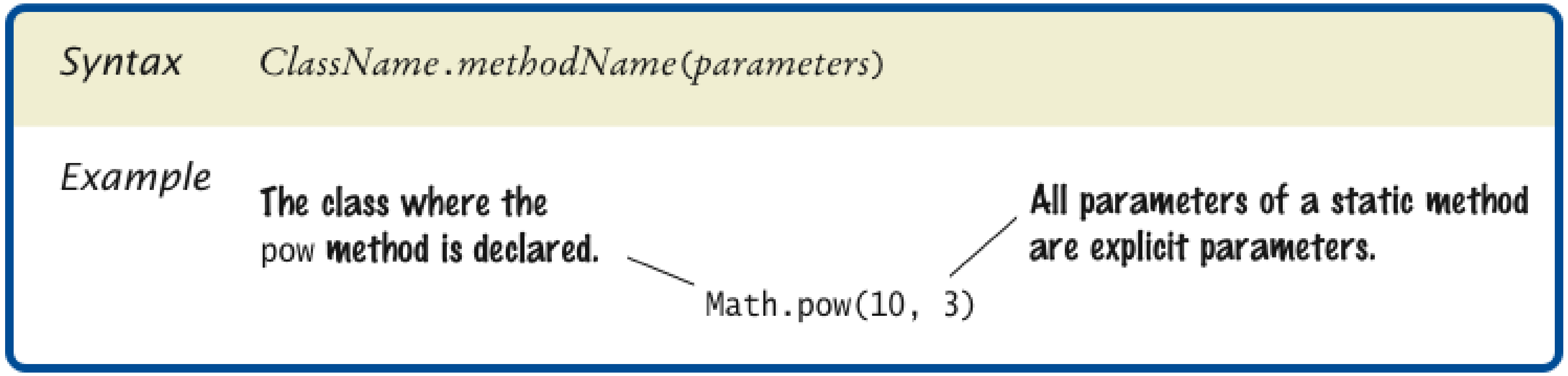
This is the type of the expression after casting.
These parentheses are a part of the cast operator.
Use parentheses here if the cast is applied to an expression with arithmetic operators.

Calling Static Methods

- A `static` method does not operate on an object

```
double x = 4;
double root = x.sqrt(); // Error
```

- Static methods are declared inside classes



Syntax *ClassName.methodName(parameters)*

Example

```
Math.pow(10, 3)
```

The class where the `pow` method is declared.

All parameters of a static method are explicit parameters.

The `String` Class

- A string is a sequence of characters
- Strings are objects of the `String` class
- A string *literal* is a sequence of characters enclosed in double quotation marks:

```
"Hello, World!"
```

- String *length* is the number of characters in the String
  - *Example:* `"Harry".length()` *is 5*
- Empty string: `""`

Concatenation

- Use the `+` operator:

```
String name = "Dave";
String message = "Hello, " + name;
// message is "Hello, Dave"
```

Concatenation in Print Statements

- Useful to reduce the number of `System.out.print` instructions:

```
System.out.print("The total is ");
System.out.println(total);
```

versus

```
System.out.println("The total is " + total);
```

Converting between Strings and Numbers

- Convert to number:

```
int n = Integer.parseInt(str);
double x = Double.parseDouble(x);
```

- Convert to string:

```
String str = "" + n;
str = Integer.toString(n);
```

Converting between Strings and Numbers

- Convert to number:

```
int n = Integer.parseInt(str);
double x = Double.parseDouble(x);
```

- Convert to string:

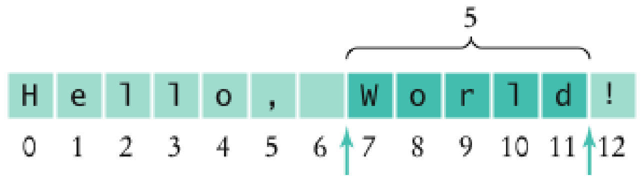
```
String str = "" + n;
str = Integer.toString(n);
```

Substrings

-
```
String greeting = "Hello, World!";
   String sub = greeting.substring(0, 5);    //sub is "Hello"
```

- Supply start and "past the end" position
- First position is at 0

```
String sub2 = greeting.substring(7, 12);
```



Reading Input

Scanner class read keyboard input in a convenient manner

```
Scanner value = new Scanner(System.in);
System.out.print("Enter quantity:");
 int quantity = value.nextInt();
```

- `nextInt` reads a `int`
- `nextDouble` reads a `double`
- `nextLine` reads a line (until user hits Enter)
- `next` reads a word (until any white space)