Konstrukcja super()

Czego się dowiesz

Czym jest i do czego służy konstrukcja super().
W jaki sposób wywoływać kod konstruktorów i metod klasy nadrzędnej w klasie pochodnej.

Słowo super w konstruktorach

W lekcji dotyczącej dziedziczenia stwierdziliśmy, że konstruktory w Javie nie są dziedziczone. Nie oznacza to jednak, że kod konstruktorów, a tym samym możliwość inicjowania pól poprzez konstruktory zupełnie znika.
Wykorzystując konstrukcję **super()** możemy wywołać konstruktor klasy nadrzędnej z odpowiednimi parametrami, a dzięki temu unikniemy powielania kodu związanego z inicjalizacją pól klasy w klasach pochodnych.
Załóżmy, że istnieje klasa Computer, a po niej dziedziczy klasa Notebook. Tym razem pokażmy, że wykorzystując pola prywatne w klasie pochodnej nadal mamy do nich dostęp za pomocą metod dostępowych.

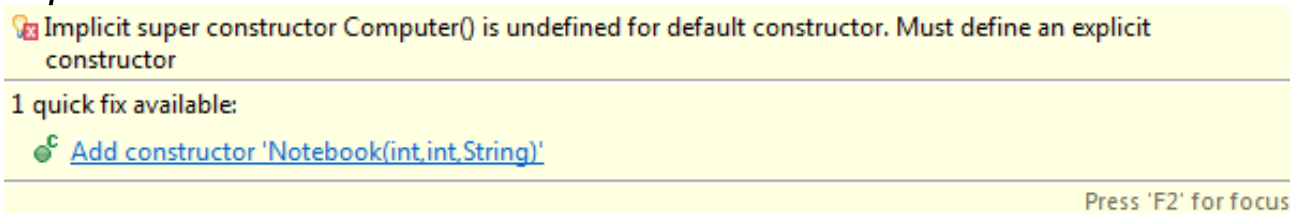
Computer.java

```
class Computer {
    private double cpuTemperature; // temp procesora
    private int ramMemory; // MB

    // konstruktor
    public Computer(double cpuTemperature, int ramMemory) {
        this.cpuTemperature = cpuTemperature;
        this.ramMemory = ramMemory;
    }

    public double getCpuTemperature() {
        return cpuTemperature;
    }

    public void setCpuTemperature(double cpuTemperature) {
        this.cpuTemperature = cpuTemperature;
    }

    public int getRamMemory() {
        return ramMemory;
    }

    public void setRamMemory(int ramMemory) {
        this.ramMemory = ramMemory;
    }
}
```

Klasa Computer przechowuje informacje o temperaturze procesora (*cpuTemperature*), a także ilości pamięci ram (*ramMemory*). Zdefiniowaliśmy dodatkowo konstruktor pozwalający zainicjować odpowiednie pola przy tworzeniu obiektu.
Stwórzmy teraz klasę Notebook, która rozszerza klasę Computer o dodatkową informację opisującą pojemności baterii:
Notebook.java

```
class Notebook extends Computer {
    private int batteryCapacity; // pojemność baterii mAh

    public int getBatteryCapacity() {
        return batteryCapacity;
    }

    public void setBatteryCapacity(int batteryCapacity) {
        this.batteryCapacity = batteryCapacity;
    }
}
```

Nie zdefiniowaliśmy żadnego konstruktora i w tej sytuacji środowisko wyświetla błąd o treści *Implicit super constructor Computer() is undefined for default constructor. Must define an explicit constructor*.



Komunikat błędu mówi o tym, że w klasie Computer nie istnieje konstruktor domyślny. Z lekcji o dziedziczeniu wiemy, że konstruktory nie są dziedziczone, więc dlaczego w klasie *Notebook* pojawia się problem? Powodem jest to, że w konstruktorze, nawet jeśli tego jawnie nie zapiszemy, wywoływana jest instrukcja super(). Służy ona do wywołania konstruktora z klasy nadrzędnej. Ponieważ w klasie Notebook nie zdefiniowaliśmy żadnego konstruktora, to jest on dla nas generowany automatycznie przez kompilator i klasa ta w rzeczywistości wygląda tak jak poniżej.

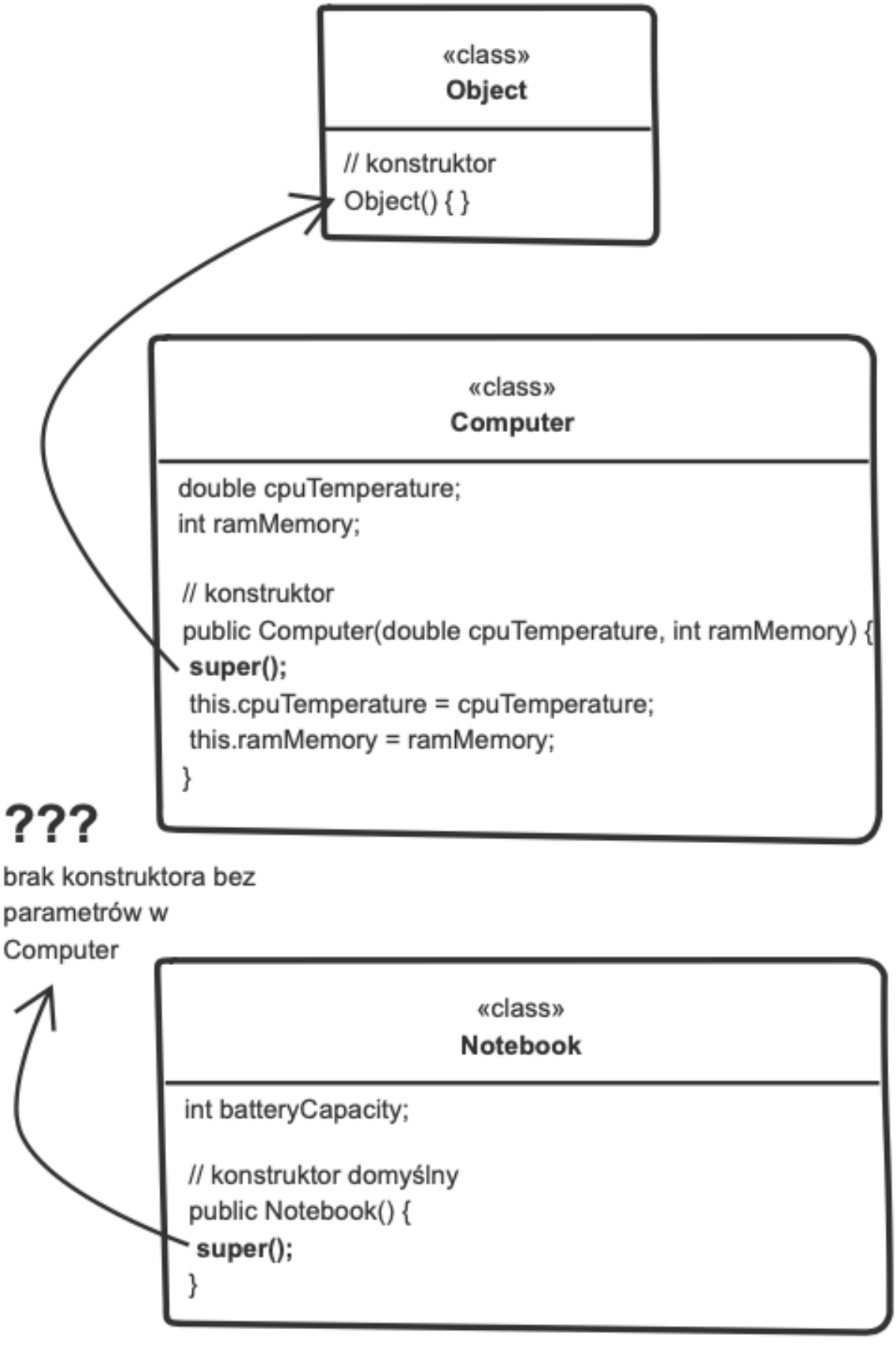
Notebook.java

```
class Notebook extends Computer {
    private int batteryCapacity; // pojemność baterii mAh

    public Notebook() {
        super();
    }

    public int getBatteryCapacity() {
        return batteryCapacity;
    }

    public void setBatteryCapacity(int batteryCapacity) {
        this.batteryCapacity = batteryCapacity;
    }
}
```

Analogicznie konstruktor w klasie *Computer* po kompilacji wygląda w rzeczywistości tak:

```
// konstruktor
public Computer(double cpuTemperature, int ramMemory) {
    super();
    this.cpuTemperature = cpuTemperature;
    this.ramMemory = ramMemory;
}
```

Instrukcja super w konstruktorze klasy Computer wywołuje konstruktor z klasy Object, natomiast instrukcja super() w konstruktorze klasy Notebook próbuje wywołać konstruktor z klasy Computer. Ponieważ Java jest językiem statycznie typowanym, to zapisanie super() z pustymi okrągłymi nawiasami oznacza wywołanie konstruktora, który nie przyjmuje żadnego argumentu. W klasie Computer stworzyliśmy konstruktor, który ma dwa parametry, a w takiej sytuacji kompilator nie wygeneruje dla nas konstruktora domyślnego. Skoro w klasie tej nie ma konstruktora domyślnego, ani jawnie nie zapisaliśmy konstruktora bez żadnych parametrów, to w klasie Notebook nie możemy takiego konstruktora z klasy Computer wywołać, bo on po prostu nie istnieje.


«class»
Object
// konstruktor
Object() { }
«class»
Computer
double cpuTemperature;
int ramMemory;
// konstruktor
public Computer(double cpuTemperature, int ramMemory) {
super();
this.cpuTemperature = cpuTemperature;
this.ramMemory = ramMemory;
}
???
brak konstruktora bez
parametrów w
Computer
«class»
Notebook
int batteryCapacity;
// konstruktor domyślny
public Notebook() {
super();
}

Istnieją dwa rozwiązania powyższego problemu:

- Jawne zapisanie konstruktora bezparametrowego w klasie Computer, dzięki czemu konstruktor domyślny z klasy Notebook będzie mógł go wywołać przez super().
- Stworzenie w klasie Notebook konstruktora, w którym jako pierwszą instrukcję jawnie wywołamy instrukcję super() i przekażemy do niej argumenty, odpowiadające tym, które przyjmuje konstruktor klasy Computer.

Drugie rozwiązanie jest bardziej praktyczne, ponieważ w klasie Notebook przydatny będzie konstruktor, który pozwoli na tworzenie obiektów tego typu w 1 wierszu, zamiast 4 (utworzenie obiektu i później ręczne ustawianie pól przez settery).

Notebook.java

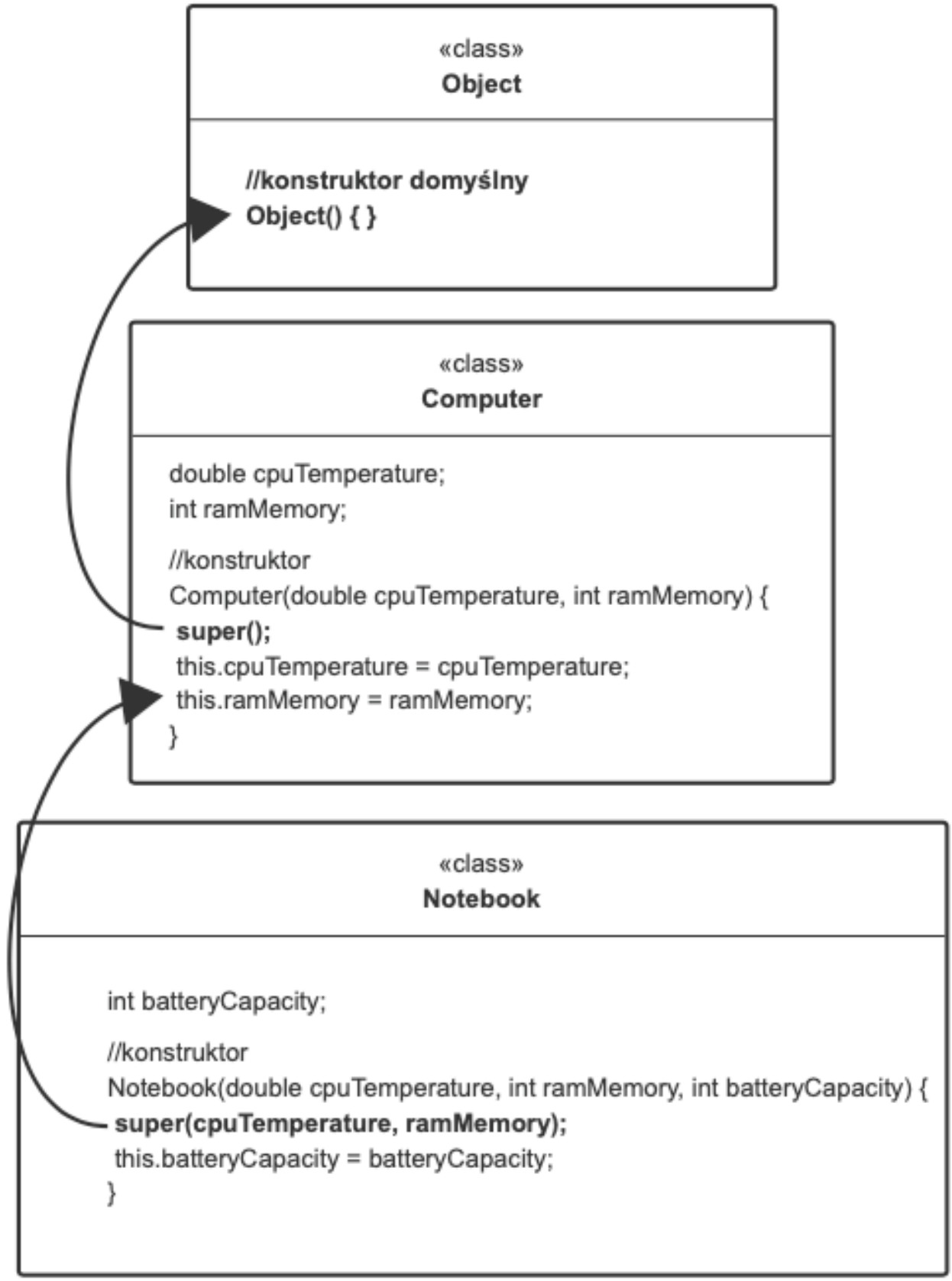
```
class Notebook extends Computer {
    private int batteryCapacity; // pojemność baterii mAh

    public Notebook(double cpuTemperature, int ramMemory, int batteryCapacity) {
        super(cpuTemperature, ramMemory); //wywołanie konstruktora z klasy Computer
        this.batteryCapacity = batteryCapacity;
    }

    public int getBatteryCapacity() {
        return batteryCapacity;
    }

    public void setBatteryCapacity(int batteryCapacity) {
        this.batteryCapacity = batteryCapacity;
    }
}
```

Zapis super(cpuTemperature, ramMemory) powoduje wywołanie konstruktora klasy nadrzędnej z podanymi parametrami, czyli naszego jedynego konstruktora w klasie Computer. Wywołanie instrukcji *super()* zawsze musi to być zapisana jako pierwsza instrukcja w konstruktorze, ale po niej mogą następować dowolne dodatkowe wywołania metod, czy przypisania pozostałych pól, tak jak w naszym przykładzie dzieje się z polem batteryCapacity.

Od teraz za każdym razem gdy utworzymy w kodzie obiekt Notebook, np.:

```
Notebook notebook1 = new Notebook(60, 8192, 4500);
```

najpierw konstruowany jest obiekt, alokowana jest pamięć dla poszczególnych pól, a następnie wywoływane są konstruktory poszczególnych klas dzięki czemu inicjowane są pola obiektu.



Słowo super w metodach

W odróżnieniu od konstruktorów, metody w Javie są dziedziczone. W lekcji o dziedziczeniu powiedzieliśmy kilka słów o ich nadpisywaniu w klasach pochodnych. Również w tym miejscu przydatna okazuje się czasami instrukcja super, jednak w nieco innej formie. Pozwala ona odwoływać się do pól, które zostały ukryte lub metod klasy nadrzędnej, które zostały nadpisane.
Jeżeli w klasie Computer dodamy metodę coolDown(), obniżającą temperaturę procesora:

Computer.java

```
class Computer {
    private double cpuTemperature; // temperatura
    private int ramMemory; // ilość pamięci w MB

    // konstruktor
    public Computer(double cpuTemperature, int ramMemory) {
        this.cpuTemperature = cpuTemperature;
        this.ramMemory = ramMemory;
    }

    public double getCpuTemperature() {
        return cpuTemperature;
    }

    public void setCpuTemperature(double cpuTemperature) {
        this.cpuTemperature = cpuTemperature;
    }

    public int getRamMemory() {
        return ramMemory;
    }

    public void setRamMemory(int ramMemory) {
        this.ramMemory = ramMemory;
    }

    public void coolDown() {
        cpuTemperature = cpuTemperature - 1;
    }
}
```

to możemy ją nadpisać w klasie Notebook. Nadpisana metoda coolDown() powinna obniżyć temperaturę procesora tak samo jak metoda w klasie Computer, ale ponieważ laptopy mają zamontowany dodatkowo specjalny układ chłodzenia, to

obniżymy temperaturę procesora o dodatkowe dwa stopnie wywołując dodatkowo metodę turboCool().

Notebook.java

```
class Notebook extends Computer {
    private int batteryCapacity; // pojemność baterii mAh

    public Notebook(double cpuTemperature, int ramMemory, int batteryCapacity) {
        super(cpuTemperature, ramMemory);
        this.batteryCapacity = batteryCapacity;
    }

    public int getBatteryCapacity() {
        return batteryCapacity;
    }

    public void setBatteryCapacity(int batteryCapacity) {
        this.batteryCapacity = batteryCapacity;
    }

    public void coolDown() {
        super.coolDown();
        turboCool();
    }

    private void turboCool() {
        setCpuTemperature(getCpuTemperature() - 2);
    }
}
```

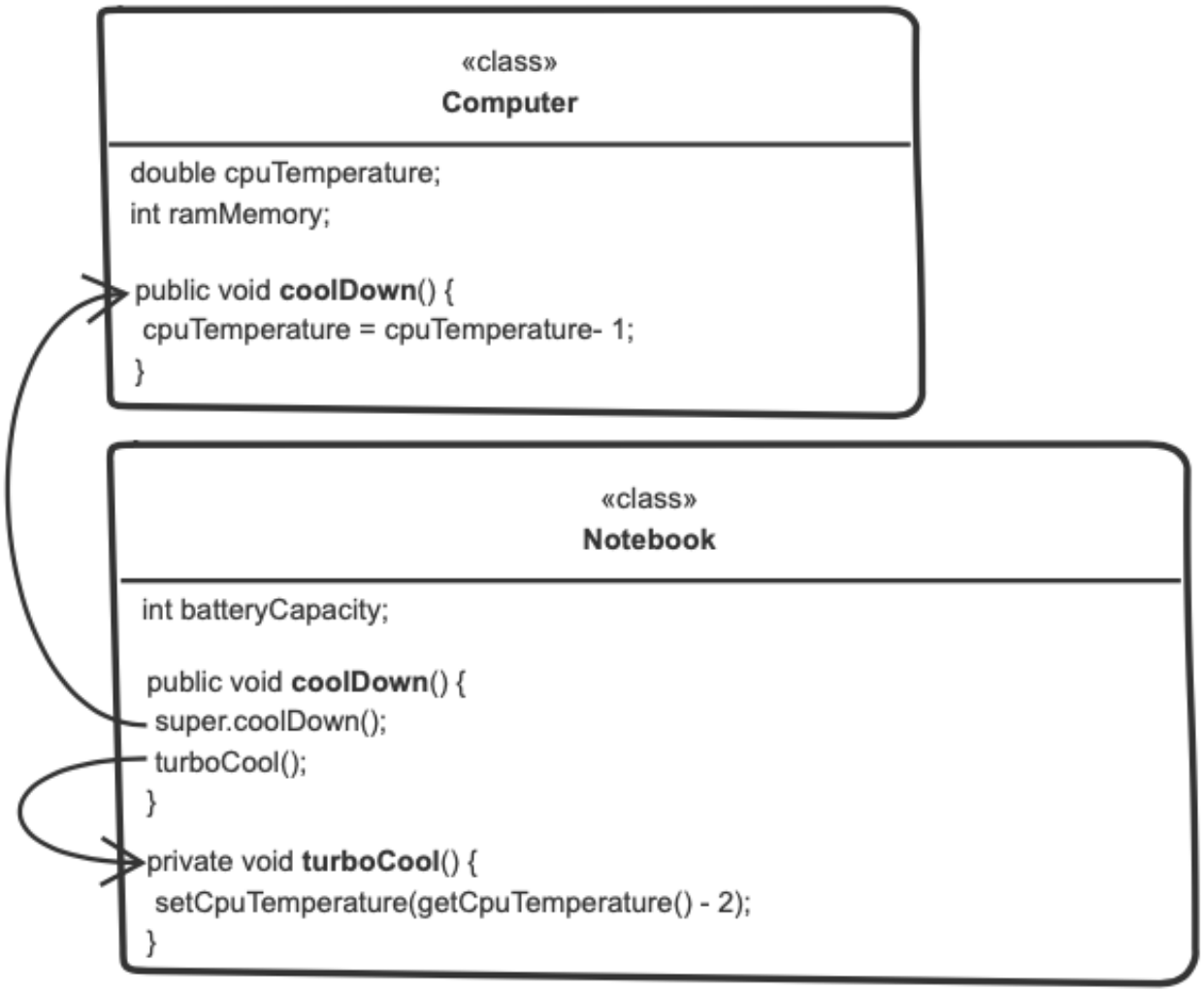
Zapis super.coolDown() powoduje wywołanie metody coolDown() z klasy nadrzędnej - Computer. Wywołanie metody *coolDown()* bez poprzedzenia jej słowem *super* oznaczałoby wywołanie metody z klasy, w której się znajdujemy (*Notebook*), co spowodowałoby zapętlenie się wywołania przez samą siebie i ostatecznie przepełnienie stosu (błąd *StackOverflowError*). W odróżnieniu od konstruktorów, wywołanie metody z klasy nadrzędnej poprzez słowo *super* nie musi być pierwszą instrukcją w metodzie.

Wywołując teraz metodę coolDown() na obiekcie typu Notebook np.:

```
Notebook notebook1 = new Notebook(60, 8192, 4500);
notebook1.coolDown();
```

najpierw wywoływana jest metoda coolDown() z klasy Computer, która obniża temperaturę o 1 stopień, a następnie wywoływana jest metoda turboCool() z klasy Notebook, która obniża temperaturę o dodatkowe 2 stopnie. Zwróć uwagę, że

w metodzie turboCool() musimy odwoływać się do temperatury poprzez settery i gettery, ponieważ jest to pole prywatne i nie mamy do niego bezpośredniego dostępu.



Metoda turboCool() w klasie Notebook także mogłaby wykorzystywać słowo super przy odwoływaniu się do setterów i getterów:

```
private void turboCool() {
    super.setCpuTemperature(super.getCpuTemperature() - 2);
}
```

W praktyce jednak nie stosuje się takiego zapisu, bo słowo super w tym przypadku nic nie wnosi.
Jak widzisz słowo *super* zachowuje się bardzo podobnie do *this*, ale o ile *this* odnosi się do klasy, w której się znajdujemy, to *super* odnosi się do klasy, po której dziedziczymy.

Zapamiętaj

Nawet jeśli tego nie zapiszesz, to pierwszą instrukcją w konstruktorze zawsze jest *super()*.
Korzystając z instrukcji *super()* możemy z konstruktora naszej klasy wywołać konstruktor z klasy nadrzędnej.
Jeżeli w klasie nadrzędnej będzie kilka konstruktorów, to wywołany będzie ten, który odpowiada przekazanym do super() argumentom.
Używając notacji *super.nazwaMetody()* możesz wywołać dowolną metodę z klasy nadrzędnej, co jest przydatne szczególnie wtedy, kiedy nadpisałeś metodę z klasy nadrzędnej w swojej klasie, a nadal potrzebujesz do niej dostępu.