Agenda

- Recap: Self-Review (Class, constructor, instance variables, methods)
- Ticket machine project: review, variables
- Operators: modulo
- Ausdrücke
- … while

Self-review

a) List the name of this method and the name and type of its parameter:

```
public void setCredits(int creditValue)
{
    credits = creditValue;
}
```

b) Write out the outer wrapping (header) of a class called Person.

c) Write out definitions for the following instance variables (fields) of the class Person:
   - A field called name of type String
   - A field of type int called age
   - A field of type String called code
   - A field called credits of type int

d) Write out a class and a constructor for a class called Module. The constructor should take a single parameter of type String called moduleCode and assign it to the field code.

Self-review

e) Write out a constructor for a class called Person. The constructor should take two parameters: a String called myName and an int called myAge. The parameters should be used to set the values of the fields name and age.

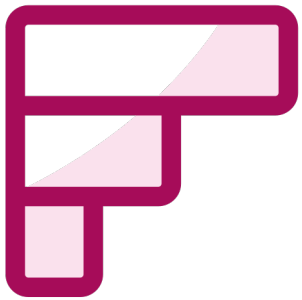
f) Write an accessor method called getName that returns the value of a field called name, whose type is String.

e) Correct the error in this method:

```
public void getAge(){
    return age;
}
```

slido

Please download and install the Slido app on all computers you use



Wie sicher fühlen Sie sich mit den Aufgaben?

ⓘ Start presenting to display the poll results on this slide.

Variables – a recap

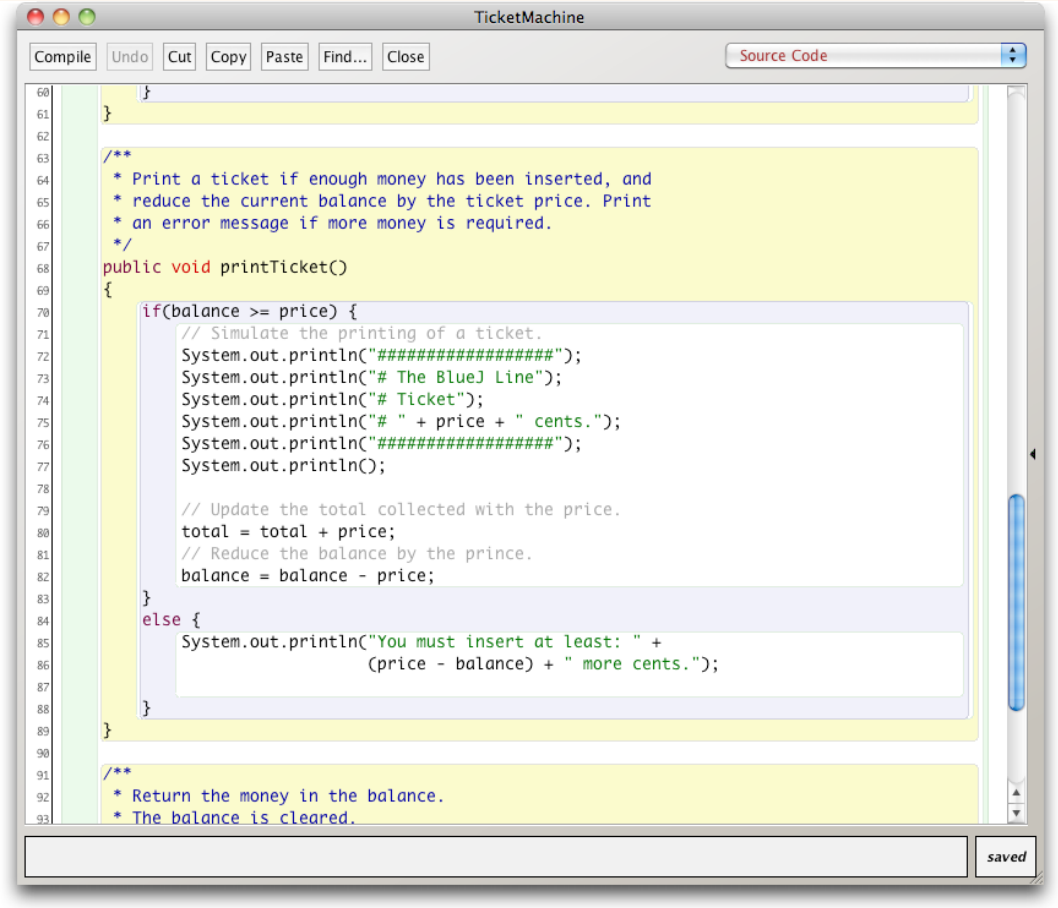
Fields are one sort of variable.

- They store values through the life of an instance.
- They are accessible throughout the class.

Parameters are another sort of variable:

- They receive values from outside the method.
- They help a method complete its task.
- Each call to the method receives a fresh set of values.
- Parameter values are short lived.

Scope highlighting



Scope and lifetime

Each block defines a new scope.

- Class, method and statement.

Scopes may be nested:

- statement block inside another block inside a method body inside a class body.

Scope is static (textual).

Lifetime is dynamic (runtime).

Local variables

Methods can define their own, *local* variables:

- Short lived, like parameters.
- The method sets their values – unlike parameters, they do not receive external values.
- Used for ‘temporary’ calculation and storage.
- They exist only as long as the method is being executed.
- They are only accessible from within the method.
- They are defined within a particular *scope*.

How do we write a method to ‘refund’ an excess balance?

Wie schreibt man eine Methode zur „Rückerstattung“ eines überschüssigen Guthabens?

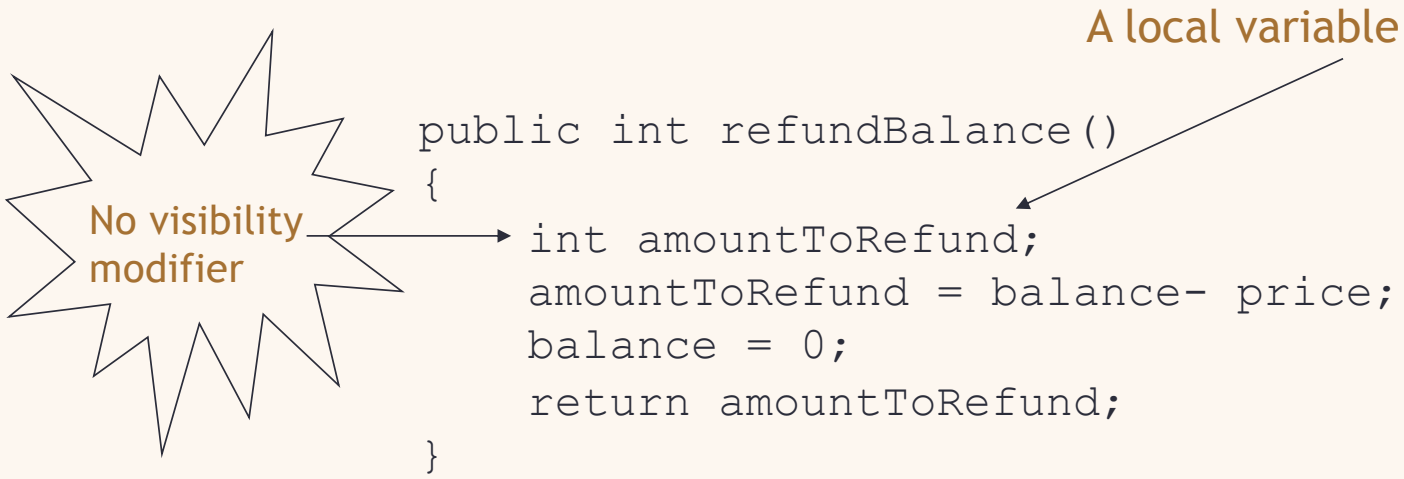
Murmelphase

Unsuccessful attempt

```
public int refundBalance()
{
    // Return the amount left.
    return balance;
    // Clear the balance.
    balance = 0;
}
```

It looks logical, but the language does not allow it.

Local variables



Scope and lifetime

The scope of a field (instance variable) is its whole class.

The lifetime of a field is the lifetime of its containing object.

The scope of a local variable is the block in which it is declared.

The lifetime of a local variable is the time of execution of the block in which it is declared.

Ausdrücke / Operatoren / Vergleiche

Integer-Arithmetik (int, short, long):	`+, -, *, /, %`
Gleitkomma-Arithmetik (float, double):	`+, -, *, /`
Boolesche Arithmetik (boolean):	`&&, ||, !`
Vergleichsoperatoren:	`==, !=, <, >, <=, >=`
Zuweisungsoperatoren:	`=, +=, -=, *=, /=, %=`
Inkrement-Operator:	`++`
Dekrement-Operator:	`--`
Bit-Operatoren:	`<<, >>>, &, |, ~, ^, ...`
Spezielle Operatoren:	`?:, (type)`

Assignment & **Expression**

Assignment statement (Zuweisung)

Expression (Ausdruck)

An expression is a combination of variables, values, **operators**, and functions that are **evaluated to produce a result**.

The outcome is either a value (with a data type) or an error.

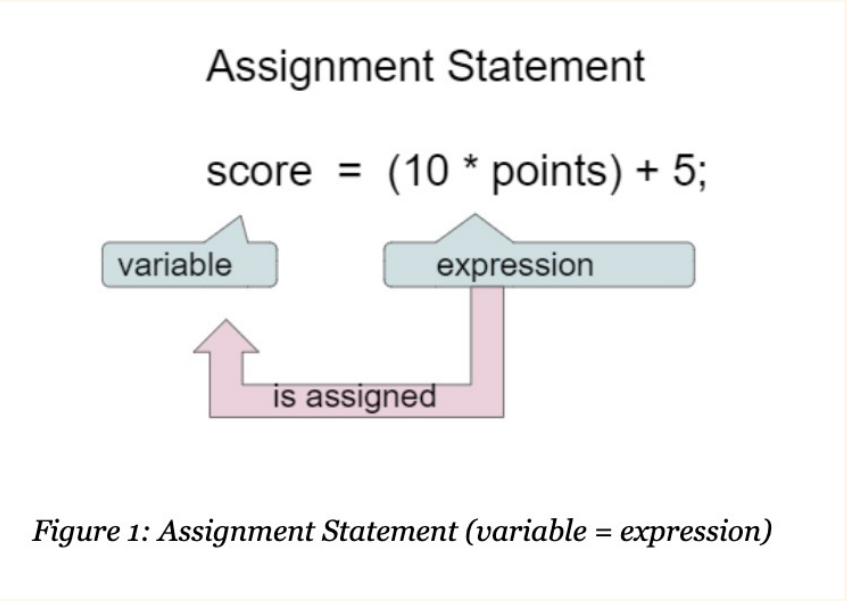


Figure 1: Assignment Statement (variable = expression)

The modulo operator

The 'division' operator (/), when applied to int operands, returns the ***result*** of an ***integer division***.

The 'modulo' operator (%) returns the ***remainder*** of an integer division.

E.g., generally:

17 / 5 gives result 3, remainder 2

In Java:

```
17 / 5 == 3
17 % 5 == 2
```

Quiz

What is the result of the expression `8 % 3`

For integer `n >= 0`, what are all possible results of:

```
n % 5
```

Can `n` be negative?

Modulo

Exp. n % 3

n	n % 3
0	0
1	1
2	2
3	0
4	1
etc.	....

Vergleichsoperatoren

- Operanden vom Typ int, short , long,float, double oder char
- Gelieferter Wert vom Typ boolean

==	Gleichheit	`4 == 5`
!=	Ungleichheit	`6 != 7`
<	Kleiner	`-2. < 0.1`
<=	Kleiner-Gleich	`3 <= 3`
>	Größer	`` `a` > `b` ``
>=	Größer-Gleich	`1 >= -1`

Vergleichsoperatoren

- Operanden vom Typ int, short , long,float, double oder char
- Gelieferter Wert vom Typ boolean

==	Gleichheit	`4 == 5  (= false)`
!=	Ungleichheit	`6 != 7  (= true)`
<	Kleiner	`-2. < 0.1  (= true)`
<=	Kleiner-Gleich	`3 <= 3  (= true)`
>	Größer	`` `a` > `b` (= false) ``
>=	Größer-Gleich	`1 >= -1  (= true)`

Ausdrücke / Operatoren / Vergleiche

- Operanden vom Typ **`boolean`**
- Gelieferter Wert vom Typ **`boolean`**

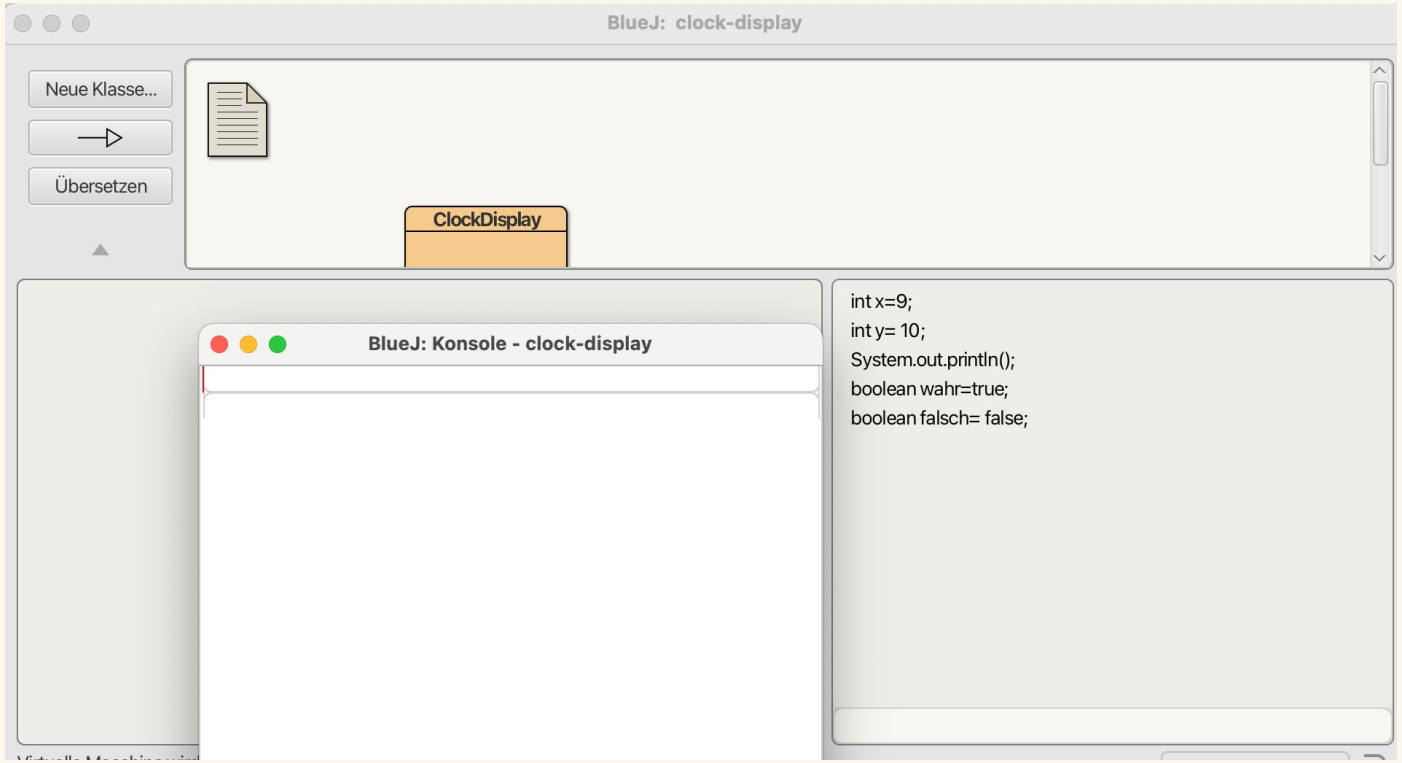
`&&`	Konjunktion	`true && false`
`||`	Disjunktion	`true || false`
`!`	Negation	`!true`

Ausdrücke / Operatoren / Vergleiche

- Operanden vom Typ **`boolean`**
- Gelieferter Wert vom Typ **`boolean`**

`&&`	Konjunktion	`true && false (= false)`
`||`	Disjunktion	`true || false (= true)`
`!`	Negation	`!true (= false)`

Playing with operators



Vergleichoperatoren

Boolean expression: Boolean expressions have only two possible values: true and false. They are commonly found controlling the choice between the two paths through a conditional statement.

https://www.linkedin.com/learning/java-grundkurs-1-sprachkonzepte-und-programmiergrundlagen/vergleichsoperatoren-nutzen?contextUrn=urn%3Ali%3AlyndaLearningPath%3A6641cb3e498ea34fbfeb9010&resume=false&u=82533698

Operatoren / Präzedenz

Liste mit absteigender Präzedenz

Postfix-Operatoren	`[], ., ++, --`
unäre Operatoren	`+, -, ~, !`
Erzeugung / Typumwandlung	`new, (type)`
Multiplikationsoperatoren	`*, /, %`
Additionsoperatoren	`+, -`
Gleichheitsoperatoren	`==, !=`
logische Und	`&&`
logisches Oder	`||`
Bedingungsoperator	`?:`
Zuweisungsoperatoren	`=, +=, *=, ...`

Abänderung der Präzedenz
durch Klammernsetzung möglich!

https://introcs.cs.princeton.edu/java/11precedence/

Ausdrücke / Operatoren / Assoziativität

bei Operatoren gleicher Präzedenz

- alle unären Operatoren von rechts errechnet (rechts-assoziativ)

- alle binären Operatoren außer den Zuweisungsoperatoren links-assoziativ
- Zuweisungsoperatoren sind rechts-assoziativ
- der ternäre Bedingungsoperator (`?:`) ist rechts-assoziativ

```
i = j *= k + 78;   ⟷   i = (j *= (k + 78));
i = j * k / 5;     ⟷   i = ((j * k) / 5);
```

Beispiele:

```
int ergebnis = 3 + 5 * 2;
System.out.println(ergebnis); // Ausgabe: 13
```

Die Multiplikation (*) hat eine höhere Präzedenz als die Addition (+).

```
int ergebnis = (3 + 5) * 2;
System.out.println(ergebnis); // Ausgabe: 16
```

Beispiele:

Relationale Operatoren (>, <) haben eine höhere Priorität als logisches UND (&&) und ODER (||).

```
boolean ergebnis = 5 > 2 && 8 > 6 || 3 < 1;
System.out.println(ergebnis); // Ausgabe: true
```

Unäre Operatoren vor binäre

```
int a = 5;
int ergebnis = -a + 10;
System.out.println(ergebnis); // Ausgabe: 5
```

Ausdrücke / Operatoren / Zuweisung

- Operanden von beliebigem Typ
- gelieferter Wert vom Typ der Operanden

```
int i = 0, j = -34; // Initialisierung
i = 45;             // Zuweisung
i = i + 3;
j = i = j + i*3;    // von rechts nach links
```

- Abkürzungen:

```
i++           ⟷   i = i+1;
i--           ⟷   i = i-1;
i += <expr>   ⟷   i = i + (<expr>)
i -= <expr>   ⟷   i = i - (<expr>)
i *= <expr>   ⟷   i = i * (<expr>)
i /= <expr>   ⟷   i = i / (<expr>)
i %= <expr>   ⟷   i = i % (<expr>)
```

Examples: Prefix vs Postfix

```
int num = 5;
int a = ++num;
int b = --num;
System.out.println(a);
System.out.println(b);
```

Prefix: 1. Ausdruck 2. Zuweisung

```
int num = 5;
int a = num++;
System.out.println(num);
System.out.println(a);
```

Postfix: 1. Zuweisung 2. Ausdruck

Examples: Prefix vs Postfix

```
int num = 5;
int a = ++num;
int b = --num;
System.out.println(a); // prints 6
System.out.println(b); // prints 5
```

```
int num = 5;
int a = num++;
System.out.println(num); // prints 6
System.out.println(a); // prints 5
```

Null

- null is a special value in Java
- Object variables and fields are initialized to null by default.
- You can test for and assign null
  - private NumberDisplay hours;
  - if(hours != null) { ... }hours = null;

slido

Please download and install the Slido app on all computers you use

Muddiest Point: Was ist Ihnen am wenigsten klar geworden?

ⓘ Start presenting to display the audience questions on this slide.

Java Schlüsselwörter

abstract	double	int	static
boolean	else	interface	super
break	extends	long	switch
byte	final	native	synchronized
case	finally	new	this
catch	float	null	throw
char	for	package	throws
class	goto	private	transient
const	if	protected	try
continue	implements	public	void
default	import	return	volatile
do	instanceof	short	while

https://www.w3schools.com/java/java_ref_keywords.asp