A Curated Inventory of Programming Language Misconceptions






Luca Chiodini

May 6, 2021 @ SI Seminar

```java
class Student {
    String firstName;
    String lastName;

    public Student(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    // ...
}
```

```java
class Student {
    String firstName;
    String lastName;

    public Student(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    // ...
}

class Main {
  public static void main(String[] args) {
    // Given two objects, check if they refer to the same student
    Student s1 = new Student("Luca", "Chiodini");
    Student s2 = new Student("Luca", "Chiodini");
    if (/* condition */) {
      System.out.println("Same student!");
    }
  }
}
```

```java
class Student {
    String firstName;
    String lastName;

    public Student(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    // ...
}

class Main {
  public static void main(String[] args) {
    // Given two objects, check if they refer to the same student
    Student s1 = new Student("Luca", "Chiodini");
    Student s2 = new Student("Luca", "Chiodini");
    if (s1 == s2) {
      System.out.println("Same student!");
    }
  }
}
```

Why so often?

Why is programming so hard?

Rainfall problem

Write a program which repeatedly reads in integers until it reads the integer 99999.

After seeing 99999, it should print out the correct average. That is, it should not count the final 99999.

Yale University

14%

86% wrong

Soloway et al.
Cognitive strategies and looping constructs: an empirical study
CACM, Nov. 1983

Why is programming so hard?

Two prerequisites

Ability to abstract

Knowledge

Why is programming so hard?

Two prerequisites

Ability to abstract

Knowledge

Knowledge for Programming

Conceptual Knowledge

Syntax and Semantics

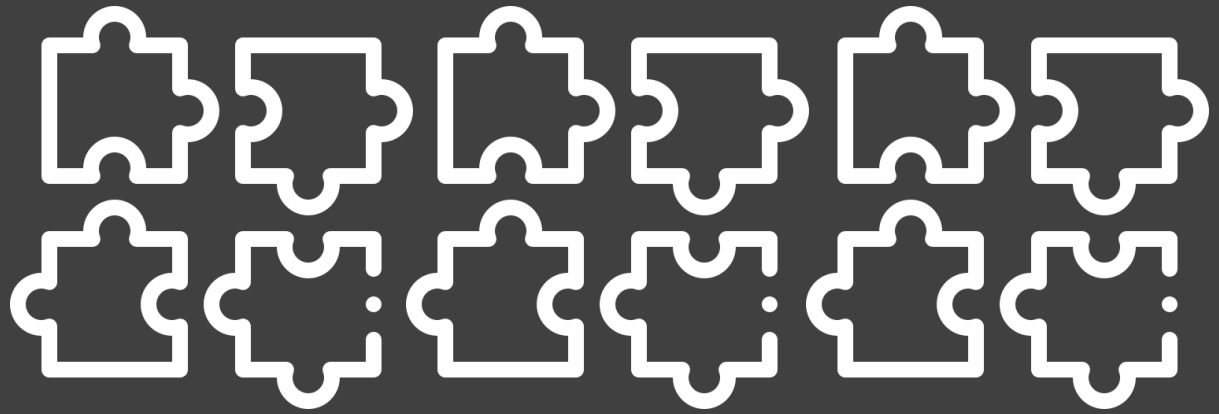
Strategic Knowledge

Pragmatics

Adapted from
Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review
Qian and Lehman, ACM Transactions on Computing Education, 2017

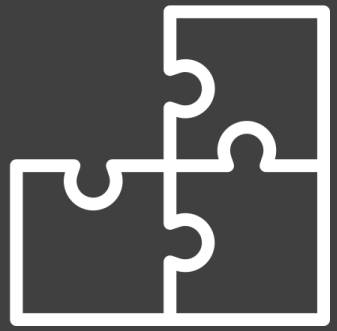
Knowledge

Knowledge-as-Elements

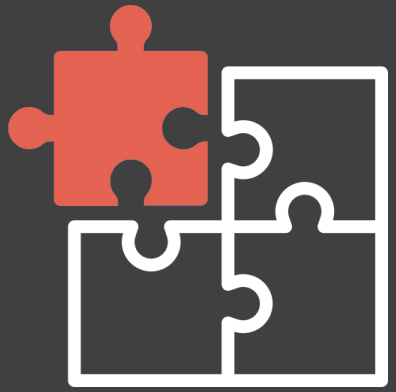


Andrea A. diSessa, 1993: **Toward an epistemology of physics**

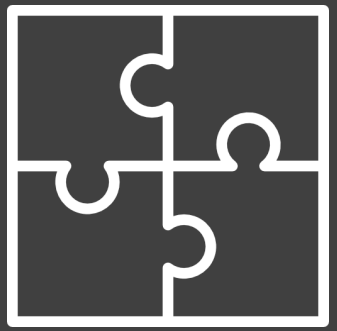
We bring Prior Knowledge



Missing conception

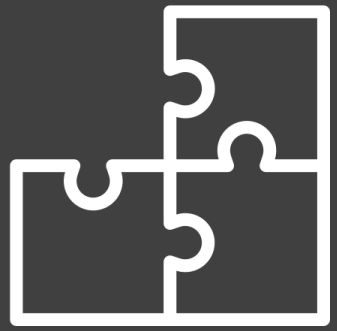


Wrong conception

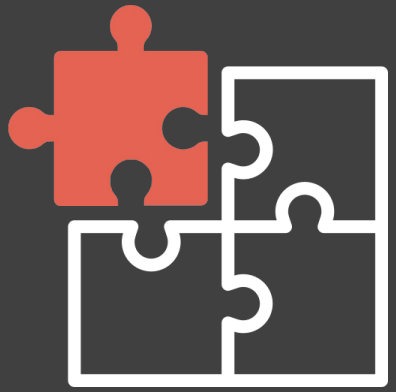


Correct conception

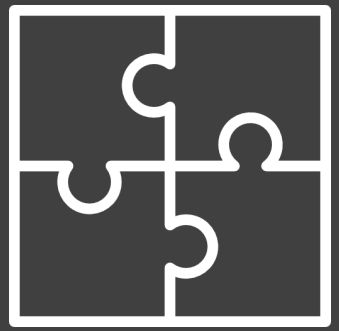
We bring Prior Knowledge



Missing conception



misconception



Correct conception

misconception

correct conception

Programming Misconceptions?

"student conceptions that produce a systematic pattern of errors"
— Smith et al.

"understandings that are deficient or inadequate for many practical programming contexts"
— Sorva

"errors in conceptual understanding"
— Qian and Lehman

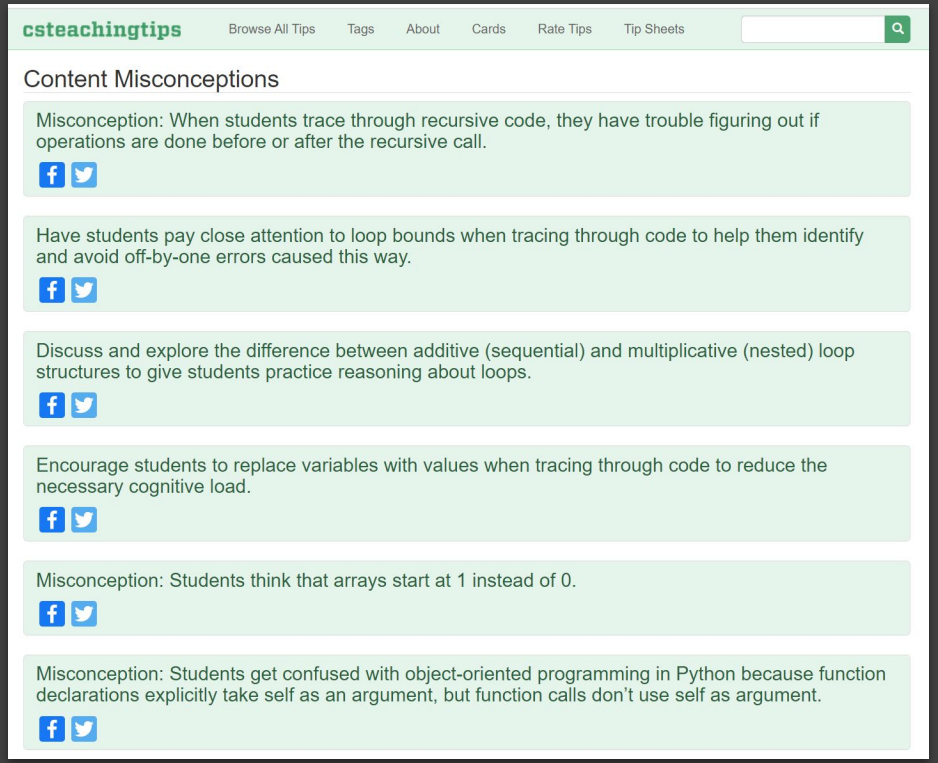
Programming Misconceptions?

Table A.1: Novice misconceptions about introductory programming content

No.	Topic	Description	Source
1	General	The computer knows the intention of the program or of a piece of code, and acts accordingly.	Pea (1986)
2	General	The computer is able to deduce the intention of the programmer.	Pea (1986)
3	General	Values are updated automatically according to a logical context.	Ragonis and Ben-Ari (2005a)
4	General	The system does not allow unreasonable operations.	Ragonis and Ben-Ari (2005a)
5	General	Difficulties understanding the lifetime of values.	du Boulay (1986)
6	General	Difficulties with telling apart the static and dynamic aspects of programs.	du Boulay (1986); Ragonis and Ben-Ari (2005a)
7	General	The machine understands English.	du Boulay (1986)
8	General	Magical parallelism: several lines of a (simple non-concurrent) program can be simultanenously active or known.	Pea (1986)
9	VarAssign	A variable can hold multiple values at a time / 'remembers' old values.	du Boulay (1986); Soloway et al. (1982); Putnam et al. (1986); Sleeman et al. (1986); Doukakis et al. (2007)
10	VarAssign	Variables always receive a particular default value upon creation.	du Boulay (1986); Samurçay (1989)
11	VarAssign	Primitive assignment works in opposite direction.	du Boulay (1986); Ma (2007); Putnam et al. (1986)
12	VarAssign	Primitive assignment works both directions (swaps).	Sleeman et al. (1986)
13	VarAssign	Limited understanding of expressions which lacks the concept of evaluation.	local

Continues on next page

Sorva's Doctoral Dissertation, 2012



Lewis's csteachingtips.org

Programming Misconceptions?

Table A.1: Novice misconceptions about introductory programming content

No.	Topic	Description	Source
1	General	The computer knows the intention of the program or of a piece of code, and acts accordingly.	Pea (1986)
2	General	The computer is able to deduce the intention of the programmer.	Pea (1986)
3	General	Values are updated automatically according to a logical context.	Ragonis and Ben-Ari (2005a)
4	General	The system does not allow unreasonable operations.	Ragonis and Ben-Ari (2005a)
5	General	Difficulties understanding the lifetime of values.	du Boulay (1986)
6	General	Difficulties with telling apart the static and dynamic aspects of programs.	du Boulay (1986); Ragonis and Ben-Ari (2005a)
7	General	The machine understands English.	du Boulay (1986)
8	General	Magical parallelism: several lines of a (simple non-concurrent) program can be simultanenously active or known.	Pea (1986)
9	VarAssign	A variable can hold multiple values at a time / 'remembers' old values.	du Boulay (1986); Soloway et al. (1982); Putnam et al. (1986); Sleeman et al. (1986); Doukakis et al. (2007)
10	VarAssign	Variables always receive a particular default value upon creation.	du Boulay (1986); Samurçay (1989)
11	VarAssign	Primitive assignment works in opposite direction.	du Boulay (1986); Ma (2007); Putnam et al. (1986)
12	VarAssign	Primitive assignment works both directions (swaps).	Sleeman et al. (1986)
13	VarAssign	Limited understanding of expressions which lacks the concept of evaluation.	local

Continues on next page

Sorva's Doctoral Dissertation, 2012

"This is a list of not only apples and oranges, but also of tomatoes and the odd dried plum."

Knowledge

Conceptual Knowledge

Syntax and Semantics

Strategic Knowledge

Pragmatics

Knowledge

Conceptual Knowledge

Syntax and Semantics

Strategic Knowledge

Pragmatics

A **programming *language* misconception** is

A **programming *language* misconception** is
a statement that can be disproved by reasoning
falsifiable statement on the syntax and/or semantics
of a programming language
about a PL

Programming *Language* Misconceptions?

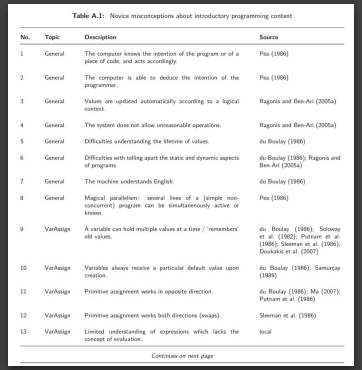


Table A.1: Novice misconceptions about introductory programming content

No.	Topic	Description	Source
1	General	The computer knows the intention of the program or of a piece of code, and acts accordingly.	Pea (1986)
2	General	The computer is able to deduce the intention of the programmer.	Pea (1986)
3	General	Values are updated automatically according to a logical context.	Ragonis and Ben-Ari (2005a)
4	General	The system does not allow unreasonable operations.	Ragonis and Ben-Ari (2005a)
5	General	Difficulties understanding the lifetime of values.	du Boulay (1986)
6	General	Difficulties with telling apart the static and dynamic aspects of programs.	du Boulay (1986); Ragonis and Ben-Ari (2005a)
7	General	The machine understands English.	du Boulay (1986)
8	General	Magical parallelism: several lines of a (simple non-concurrent) program can be simultaneously active or known.	Pea (1986)
9	VarAssign	A variable can hold multiple values at a time / 'remembers' old values.	du Boulay (1986); Soloway et al. (1982); Putnam et al. (1986); Sleeman et al. (1986); Doukakis et al. (2007)
10	VarAssign	Variables always receive a particular default value upon creation.	du Boulay (1986); Samurçay (1989)
11	VarAssign	Primitive assignment works in opposite direction.	du Boulay (1986); Ma (2007); Putnam et al. (1986)
12	VarAssign	Primitive assignment works both directions (swaps).	Sleeman et al. (1986)
13	VarAssign	Limited understanding of expressions which lacks the concept of evaluation.	local

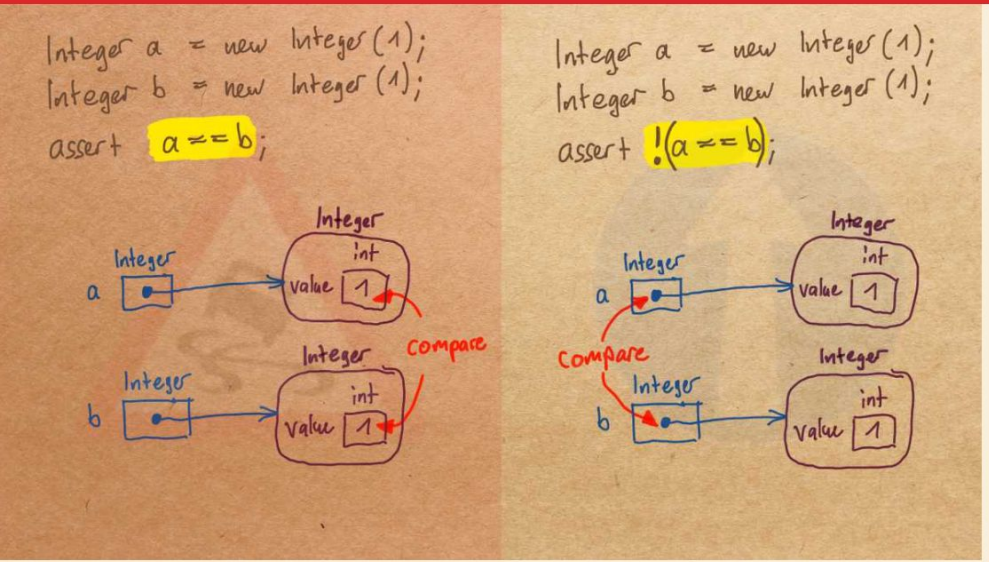
Continues on next page

82	ObjClass	A set (such as "team" or "the species of birds") cannot be a class.	Teif and Hazzan (2006)

85	ObjClass	Difficulties understanding the empty constructor.	Ragonis and Ben-Ari (2005a)

progmiscon.org

EqualityOperatorComparesObjectsValues



Incorrect

o==p compares the objects referred to by variables o and p

Correct

o==p compares the references stored in the variables o and p

Correction

Here is what's right.

The Java operator `==` checks for *reference equality*, that is it checks whether its left and right operands **refer to the same object**.

```
Integer a = new Integer(1);
Integer b = new Integer(1);
```

Language

Java

jls13 15.21 Equality Operators

Concepts

Equality, Reference, Object, Value, Operator, Expression

Expressible In

StackHeapGlobalDiagram
two reference variables and two heap objects

Related Misconceptions

EqualsComparesReferences
Dual

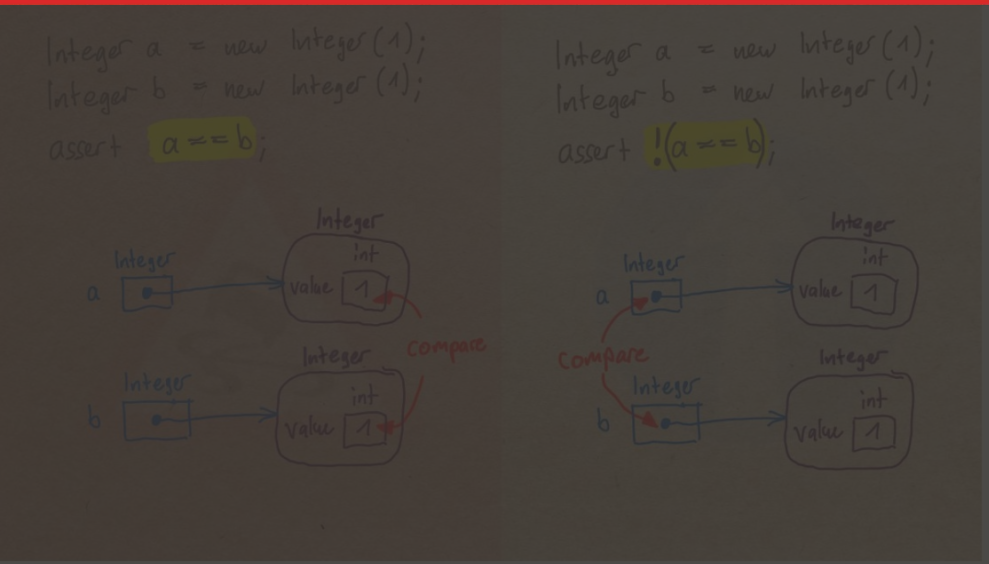
AssignmentCopiesObject
Parallel (also about reference vs. value)

ObjectAsParameterIsCopied
Parallel (also about reference vs. value)

Other Languages

JavaScript EqualityOperatorComparesObjectsValues

EqualityOperatorComparesObjectsValues



Incorrect

o==p compares the objects referred to by variables o and p

Correct

o==p compares the references stored in the variables o and p

Correction

Here is what's right.

The Java operator == checks for *reference equality*, that is it checks whether its left and right operands **refer to the same object**.

```
Integer a = new Integer(1);
Integer b = new Integer(1);
```

Language

Java

jls13 15.21 Equality Operators

Concepts

Equality, Reference, Object, Value, Operator, Expression

Expressible In

StackHeapGlobalDiagram
two reference variables and two heap objects

Related Misconceptions

EqualsComparesReferences
Dual

AssignmentCopiesObject
Parallel (also about reference vs. value)

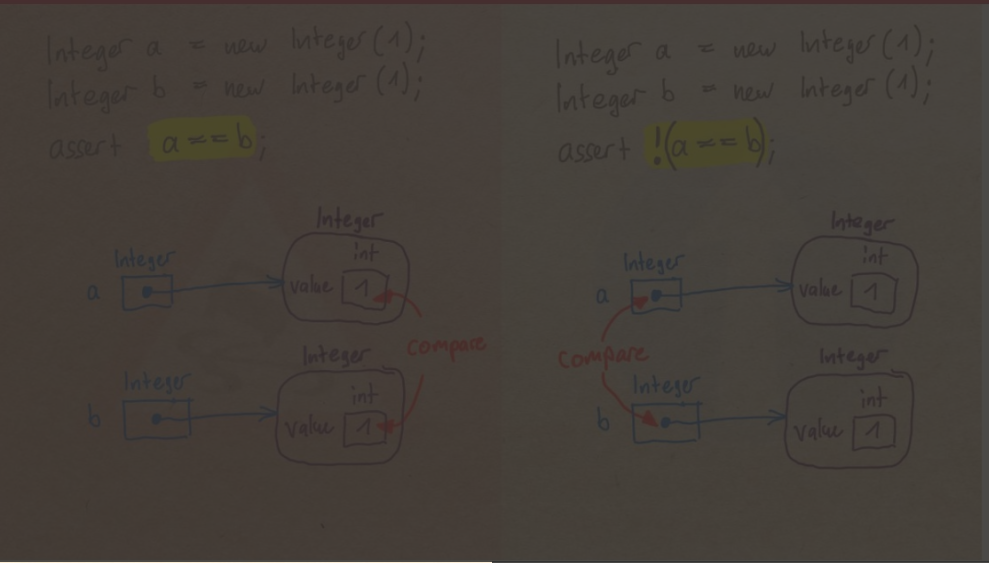
ObjectAsParameterIsCopied
Parallel (also about reference vs. value)

Other Languages

JavaScript EqualityOperatorComparesObjectsValues

Memorable Name

EqualityOperatorComparesObjectsValues



Language

Java

jls13 15.21 Equality Operators

Concepts

Equality · Reference · Object · Value · Operator · Expression

Expressible In

StackHeapGlobalDiagram
two reference variables and two heap objects

Related Misconceptions

EqualsComparesReferences
Dual

AssignmentCopiesObject
Parallel (also about reference vs. value)

ObjectAsParameterIsCopied
Parallel (also about reference vs. value)

Other Languages

JavaScript EqualityOperatorComparesObjectsValues

Incorrect

o==p compares the objects referred to by variables o and p

Correct

o==p compares the references stored in the variables o and p

Correction

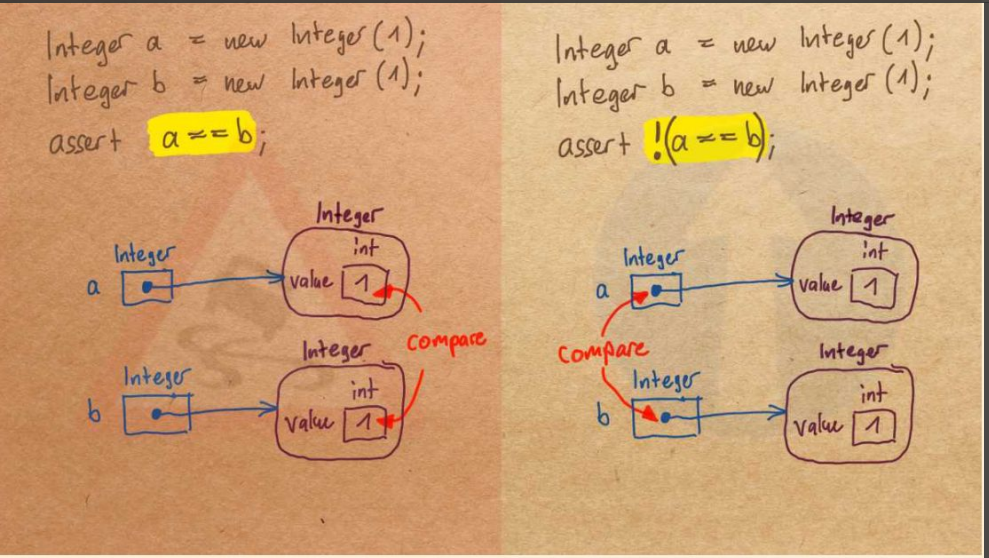
Here is what's right.

The Java operator == checks for *reference equality*, that is it checks whether its left and right operands **refer to the same object**.

```
Integer a = new Integer(1);
Integer b = new Integer(1);
```

Statement

EqualityOperatorComparesObjectsValues



Incorrect

o==p compares the objects referred to by variables o and p

Correct

o==p compares the references stored in the variables o and p

Correction

Here is what's right.

The Java operator == checks for *reference equality*, that is it checks whether its left and right operands refer to the same object.

```
Integer a = new Integer(1);
Integer b = new Integer(1);
```

Language

Java

jls13 15.21 Equality Operators

Concepts

Equality · Reference · Object · Value · Operator · Expression

Expressible In

StackHeapGlobalDiagram
two reference variables and two heap objects

Related Misconceptions

EqualsComparesReferences
Dual

AssignmentCopiesObject
Parallel (also about reference vs. value)

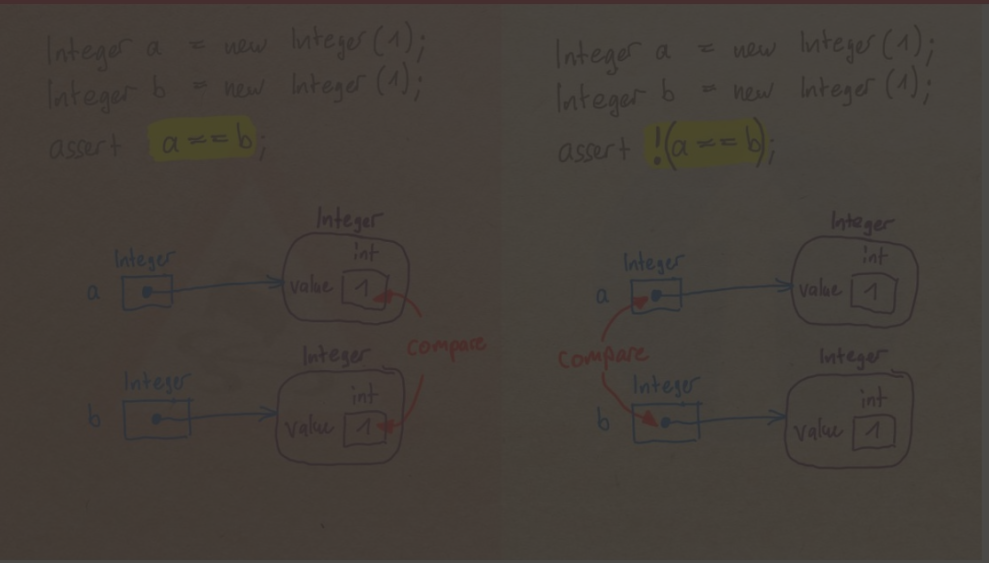
ObjectAsParameterIsCopied
Parallel (also about reference vs. value)

Other Languages

JavaScript EqualityOperatorComparesObjectsValues

Incorrect
vs
Correct
(Picture + Text)

EqualityOperatorComparesObjectsValues



Incorrect

o==p compares the objects referred to by variables o and p

Correct

o==p compares the references stored in the variables o and p

Correction

Here is what's right.

The Java operator `==` checks for *reference equality*, that is it checks whether its left and right operands **refer to the same object**.

```
Integer a = new Integer(1);
Integer b = new Integer(1);
```

Language

Java

jls13 15.21 Equality Operators

Concepts

Equality, Reference, Object, Value, Operator, Expression

Expressible In

StackHeapGlobalDiagram
two reference variables and two heap objects

Related Misconceptions

EqualsComparesReferences
Dual

AssignmentCopiesObject
Parallel (also about reference vs. value)

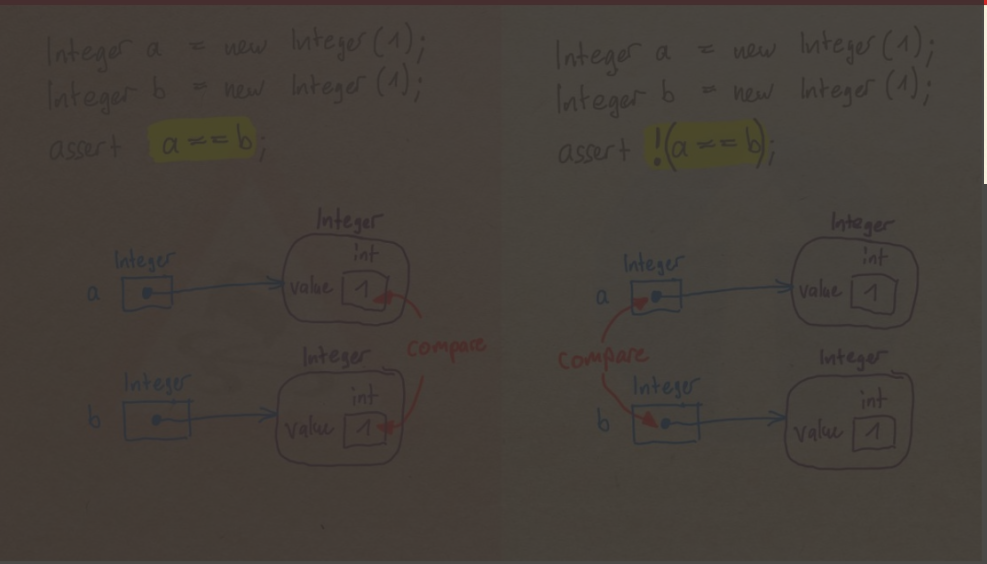
ObjectAsParameterIsCopied
Parallel (also about reference vs. value)

Other Languages

JavaScript EqualityOperatorComparesObjectsValues

Correction

EqualityOperatorComparesObjectsValues



Incorrect

o==p compares the objects referred to by variables o and p

Correct

o==p compares the references stored in the variables o and p

Correction

Here is what's right.

The Java operator == checks for *reference equality*, that is it checks whether its left and right operands **refer to the same object**.

```
Integer a = new Integer(1);
Integer b = new Integer(1);
```

Language

Java

jls13 15.21 Equality Operators

Concepts

Equality · Reference · Object · Value · Operator · Expression

Expressible In

StackHeapGlobalDiagram
two reference variables and two heap objects

Related Misconceptions

EqualsComparesReferences
Dual

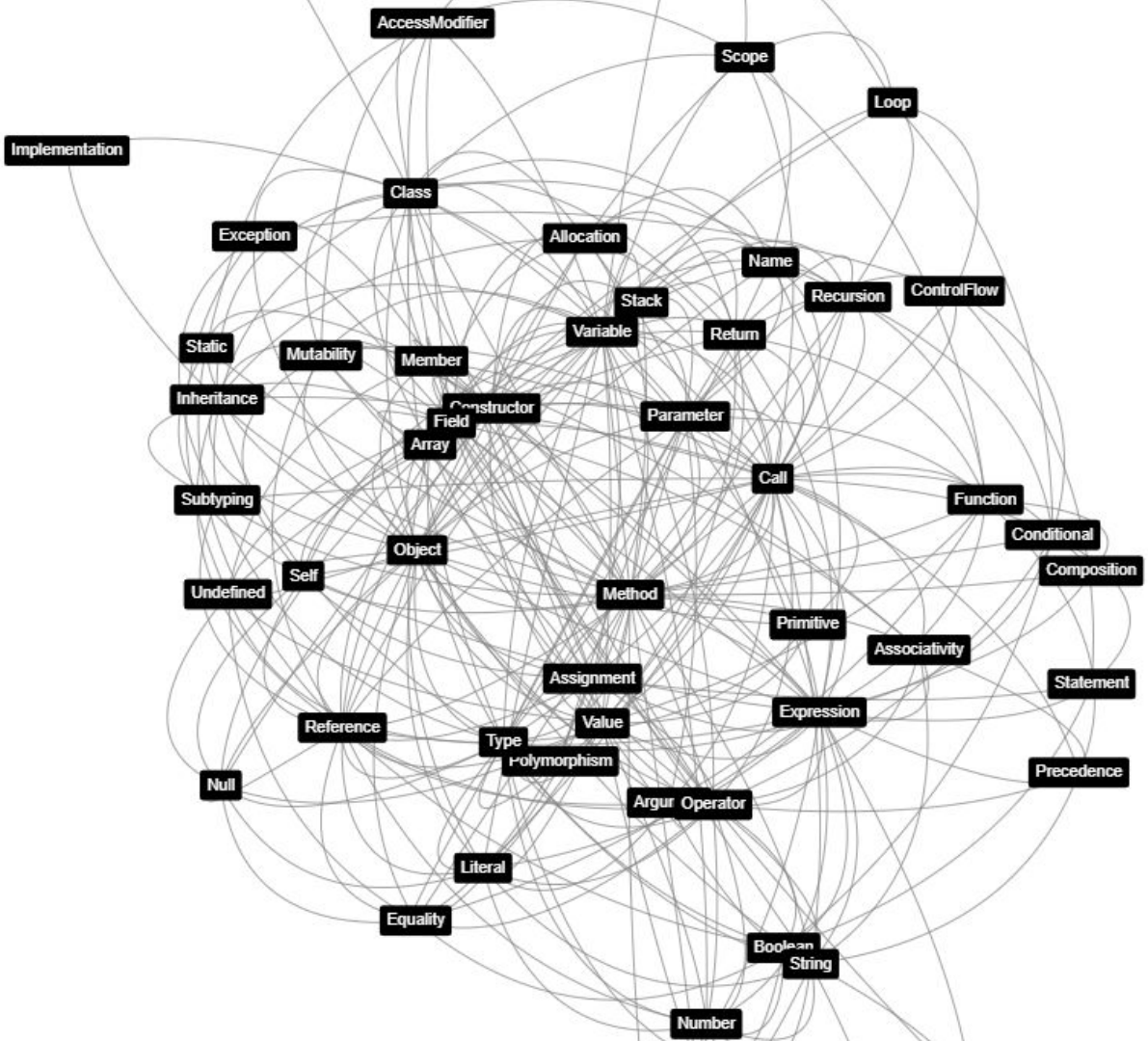
AssignmentCopiesObject
Parallel (also about reference vs. value)

ObjectAsParameterIsCopied
Parallel (also about reference vs. value)

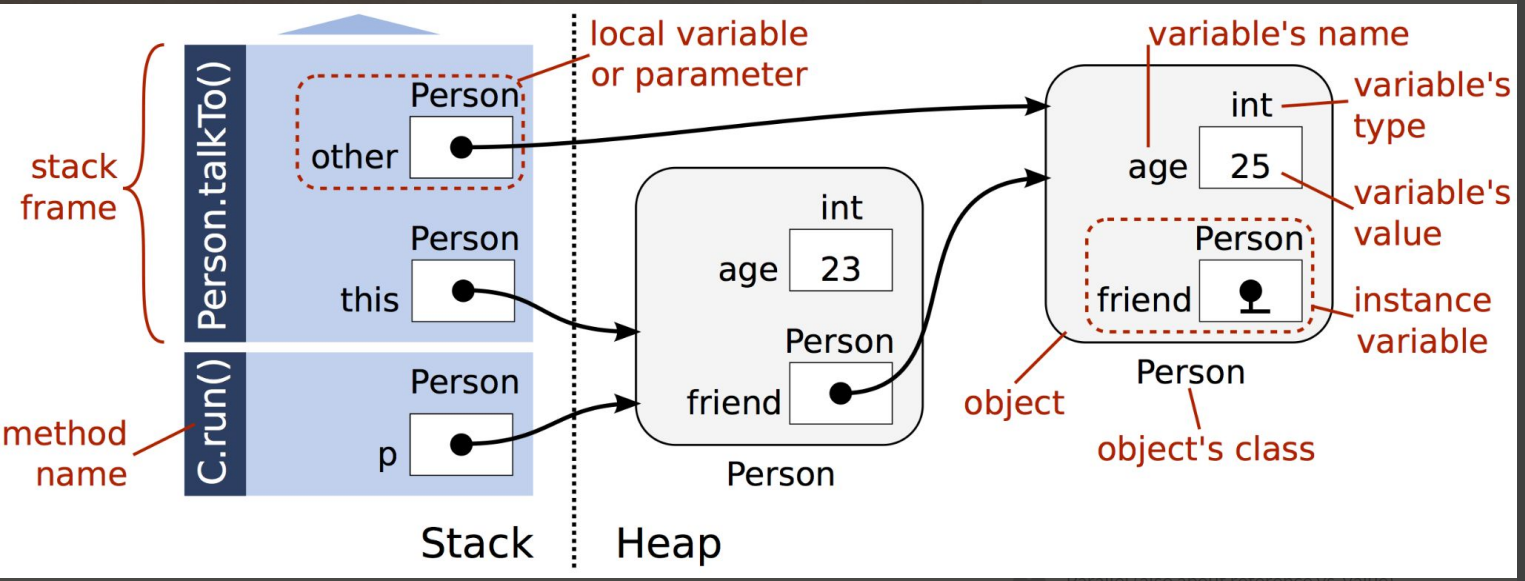
Other Languages

JavaScript EqualityOperatorComparesObjectsValues

Authoritative Source for Truth: PL specification

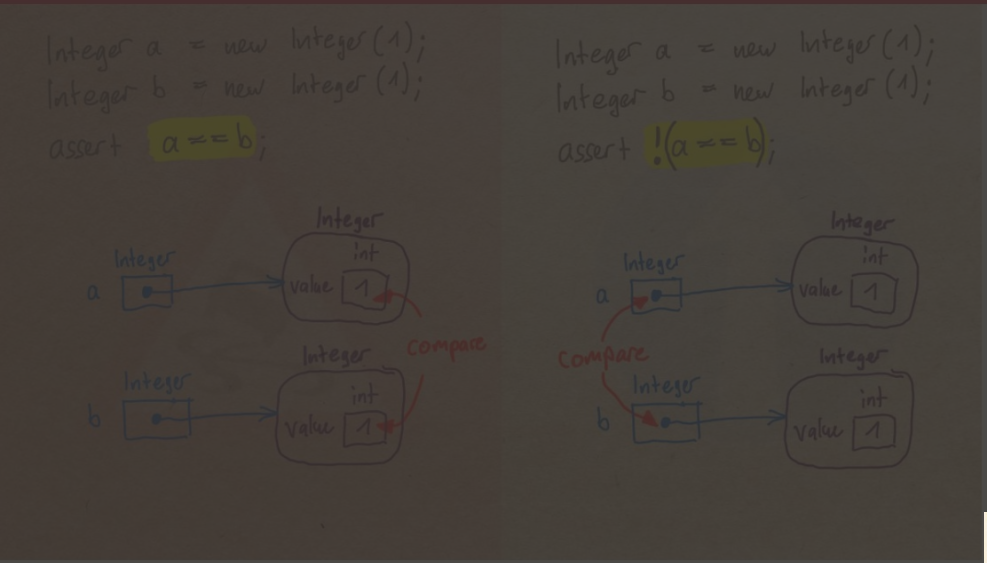

AccessModifier
Scope
Loop
Implementation
Class
Exception
Allocation
Name
Recursion
ControlFlow
Stack
Variable
Return
Static
Mutability
Member
Inheritance
Constructor
Field
Parameter
Array
Call
Subtyping
Function
Conditional
Object
Composition
Self
Undefined
Method
Primitive
Associativity
Assignment
Statement
Reference
Value
Expression
Type
Polymorphism
Precedence
Null
Operator
Literal
Equality
String
Number

Concepts



Notional Machines

EqualityOperatorComparesObjectsValues



Incorrect

o==p compares the objects referred to by variables o and p

Correct

o==p compares the references stored in the variables o and p

Correction

Here is what's right.

The Java operator == checks for *reference equality*, that is it checks whether its left and right operands **refer to the same object**.

```
Integer a = new Integer(1);
Integer b = new Integer(1);
```

Language

Java

jls13 15.21 Equality Operators

Concepts

Equality, Reference, Object, Value, Operator, Expression

Expressible In

StackHeapGlobalDiagram
two reference variables and two heap objects

Related Misconceptions

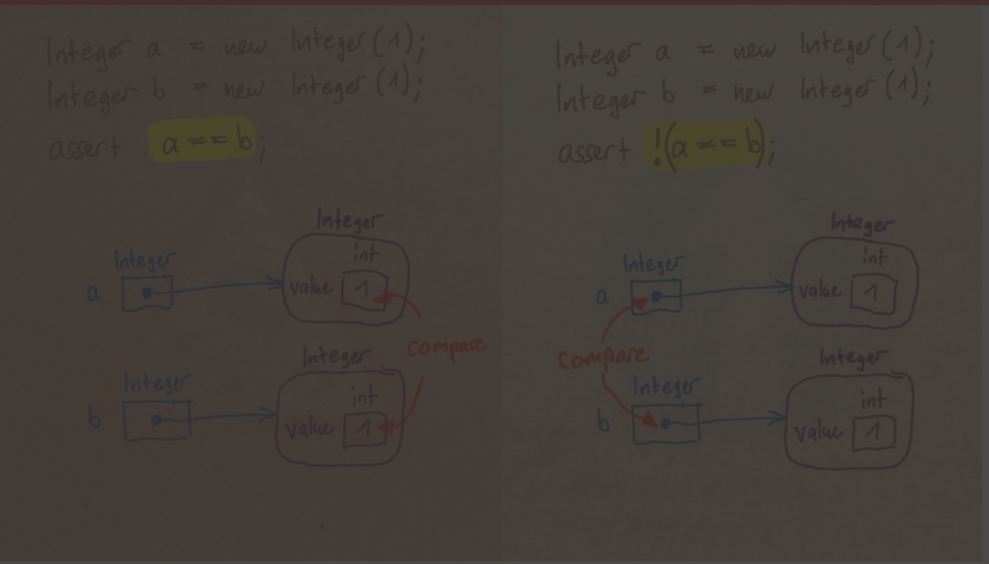
- EqualsComparesReferences
  Dual
- AssignmentCopiesObject
  Parallel (also about reference vs. value)
- ObjectAsParameterIsCopied
  Parallel (also about reference vs. value)

Other Languages

JavaScript EqualityOperatorComparesObjectsValues

Related Misconceptions

EqualityOperatorComparesObjectsValues



Incorrect

o==p compares the objects referred to by variables o and p

Correct

o==p compares the references stored in the variables o and p

Correction

Here is what's right.

The Java operator == checks for *reference equality*, that is it checks whether its left and right operands **refer to the same object**.

```
Integer a = new Integer(1);
Integer b = new Integer(1);
```

Language

Java

jls13 15.21 Equality Operators

Concepts

Equality, Reference, Object, Value, Operator, Expression

Expressible In

StackHeapGlobalDiagram
two reference variables and two heap objects

Related Misconceptions

EqualsComparesReferences
Dual

AssignmentCopiesObject
Parallel (also about reference vs. value)

ObjectAsParameterIsCopied
Parallel (also about reference vs. value)

Other Languages

JavaScript EqualityOperatorComparesObjectsValues

Same in other PLs

EqualityOperatorComparesObjectsValues

Incorrect

o==p compares the objects referred to by variables o and p

Correct

o==p compares the references stored in the variables o and p

Origin

Where could this misconception come from?

When they start to learn how to program, students usually only deal with primitive values, that are indeed directly comparable using the `==` operator.

That knowledge might be improperly transferred to complex data structures and/or objects.

Symptoms

How do you know your students might have this misconception?

Students use the `==` operator to compare objects' state.

They don't know the existence of an `equals` method in the `Object` class, they have no clue about when it should be used, and they never implement an `equals` method for the classes they write.

Value

How can you build on this misconception?

This misconception is a good opportunity to discuss the difference between the general concepts of **structural equality** (equal objects, i.e., two objects with equivalent state) and **referential equality** (same object).

Some languages, e.g., Kotlin, provide separate operators for the two kinds of equality. Java only provides an operator for reference equality (`==`) and expects developers to call the `equals(Object)` method to test for structural equality.

Language

Java

jls13 15.21 Equality Operators

Concepts

Equality, Reference, Object, Value, Operator, Expression

Expressible In

StackHeapGlobalDiagram
two reference variables and two heap objects

Related Misconceptions

EqualsComparesReferences
Dual

AssignmentCopiesObject
Parallel (also about reference vs. value)

ObjectAsParameterIsCopied
Parallel (also about reference vs. value)

Other Languages

JavaScript EqualityOperatorComparesObjectsValues

Origin

EqualityOperatorComparesObjectsValues

Incorrect

o==p compares the objects referred to by variables o and p

Correct

o==p compares the references stored in the variables o and p

Origin

Where could this misconception come from?

When they start to learn how to program, students usually only deal with primitive values, that are indeed directly comparable using the `==` operator.

That knowledge might be improperly transferred to complex data structures and/or objects.

Symptoms

How do you know your students might have this misconception?

Students use the `==` operator to compare objects' state.

They don't know the existence of an `equals` method in the `Object` class, they have no clue about when it should be used, and they never implement an `equals` method for the classes they write.

Value

How can you build on this misconception?

This misconception is a good opportunity to discuss the difference between the general concepts of **structural equality** (equal objects, i.e., two objects with equivalent state) and **referential equality** (same object).

Some languages, e.g., Kotlin, provide separate operators for the two kinds of equality. Java only provides an operator for reference equality (`==`) and expects developers to call the `equals(Object)` method to test for structural equality.

Language

Java

jls13 15.21 Equality Operators

Concepts

Equality | Reference | Object | Value | Operator | Expression

Expressible In

StackHeapGlobalDiagram
two reference variables and two heap objects

Related Misconceptions

EqualsComparesReferences
Dual

AssignmentCopiesObject
Parallel (also about reference vs. value)

ObjectAsParameterIsCopied
Parallel (also about reference vs. value)

Other Languages

JavaScript EqualityOperatorComparesObjectsValues

Symptoms

ConstReferenceImpliesImmutability

Incorrect

An object referred to by a const variable is an immutable object

Correct

An object referred to by a const variable can be a mutable object

Symptoms

How do you know your students might have this misconception?

Watch a real student who might have this misconception solving a programming exercise!

Language

JavaScript

ecma11 13.3.1 Let and Const Declarations

Concepts

Mutability Reference

Other Languages

Java FinalReferenceImpliesImmutability

Symptoms

ConstReferenceImpliesImmutability

Incorrect

An object referred to by a const variable is an immutable object

Correct

An object referred to by a const variable can be a mutable object

Symptoms

How do you know your students might have this misconception?

Watch a real student who might have this misconception solving a programming exercise!

Language

JavaScript

ecma11 13.3.1 Let and Const Declarations

Concepts

Mutability Reference

Other Languages

Java FinalReferenceImpliesImmutability

Symptoms

ParenthesesOnlyIfArgument

Incorrect

() are optional for method calls without arguments

Correct

() are mandatory even for method calls without arguments

Correction

Here is what's right.

In Java parentheses are required to distinguish method calls (`o.m()`) from field accesses (`o.f`).

When calling any method, even a method without arguments, parentheses are required.

Value

How can you build on this misconception?

At first this misconception may look purely superficial. However, it can be surprisingly deep.

In languages where functions are values, it is essential to understand the difference between accessing the function as a value (`f`), and invoking a function (`f()`). With the addition of lambdas and method references to Java, Java now also supports treating functions as values. However, in Java—unlike in other languages with first-class functions—to invoke such functions, one needs to use the traditional method invocation mechanism to call the single method provided by the functional interface that represents the function's type:

```
Function<Integer,Integer> f = ...;`
int y = f.apply(x); // instead of f(x) used in other languages
```

An additional potentially valuable aspect of this misconception is that it might indicate that a student sees methods and fields as something similar. And indeed, in some languages (like JavaScript) objects are seen primarily as a general map from keys to values (sometimes called a "hash"). Some entries in that map can be seen as "methods", because their values are functions.

Language

Java

jls13 15.12 Method Invocation Expressions

jls13 15.8.5 Parenthesized Expressions

Concepts

Method | Call | Expression

Expressible In

ExpressionAsTree
expression involving call of method without arguments

Related Misconceptions

ReturnCall
Also about need for parentheses

Literature References

altadmri37MillionCompilations2015
J Forgetting parentheses after a method call

hristovaIdentifyingCorrectingJava2003
10 Forgetting parentheses after a method call

Value

ParenthesesOnlyIfArgument

Incorrect

() are optional for method calls without arguments

Correct

() are mandatory even for method calls without arguments

Correction

Here is what's right.

In Java parentheses are required to distinguish method calls (`o.m()`) from field accesses (`o.f`).

When calling any method, even a method without arguments, parentheses are required.

Value

How can you build on this misconception?

At first this misconception may look purely superficial. However, it can be surprisingly deep.

In languages where functions are values, it is essential to understand the difference between accessing the function as a value (`f`), and invoking a function (`f()`). With the addition of lambdas and method references to Java, Java now also supports treating functions as values. However, in Java—unlike in other languages with first-class functions—to invoke such functions, one needs to use the traditional method invocation mechanism to call the single method provided by the functional interface that represents the function's type:

```
Function<Integer,Integer> f = ...;`
int y = f.apply(x); // instead of f(x) used in other languages
```

An additional potentially valuable aspect of this misconception is that it might indicate that a student sees methods and fields as something similar. And indeed, in some languages (like JavaScript) objects are seen primarily as a general map from keys to values (sometimes called a "hash"). Some entries in that map can be seen as "methods", because their values are functions.

Language

Java

jls13 15.12 Method Invocation Expressions

jls13 15.8.5 Parenthesized Expressions

Concepts

Method Call Expression

Expressible In

ExpressionAsTree
expression involving call of method without arguments

Related Misconceptions

ReturnCall
Also about need for parentheses

Literature References

altadmri37MillionCompilations2015
J Forgetting parentheses after a method call

hristovaIdentifyingCorrectingJava2003
10 Forgetting parentheses after a method call

Value

ParenthesesOnlyIfArgument

Incorrect

() are optional for method calls without arguments

Correct

() are mandatory even for method calls without arguments

Correction

Here is what's right.

In Java parentheses are required to distinguish method calls (`o.m()`) from field accesses (`o.f`).

When calling any method, even a method without arguments, parentheses are required.

Value

How can you build on this misconception?

At first this misconception may look purely superficial. However, it can be surprisingly deep.

In languages where functions are values, it is essential to understand the difference between accessing the function as a value (`f`), and invoking a function (`f()`). With the addition of lambdas and method references to Java, Java now also supports treating functions as values. However, in Java—unlike in other languages with first-class functions—to invoke such functions, one needs to use the traditional method invocation mechanism to call the single method provided by the functional interface that represents the function's type:

```
Function<Integer,Integer> f = ...;
int y = f.apply(x); // instead of f(x) used in other languages
```

An additional potentially valuable aspect of this misconception is that it might indicate that a student sees methods and fields as something similar. And indeed, in some languages (like JavaScript) objects are seen primarily as a general map from keys to values (sometimes called a "hash"). Some entries in that map can be seen as "methods", because their values are functions.

Language

Java

jls13 15.12 Method Invocation Expressions

jls13 15.8.5 Parenthesized Expressions

Concepts

Method Call Expression

Expressible In

ExpressionAsTree
expression involving call of method without arguments

Related Misconceptions

ReturnCall
Also about need for parentheses

Literature References

altadmri37MillionCompilations2015
J Forgetting parentheses after a method call

hristovaIdentifyingCorrectingJava2003
10 Forgetting parentheses after a method call

Literature References

Authors	Year	Name	Statement	Correction	Concepts	PL Spec.	Papers	Textbooks	Rel. Miscon.	In other PLs	Symptoms	Observations	Origins	Value
Bayman and Mayer [2]	1983	◐	●		◐						●			◐
du Boulay [7]	1986		●		●		◐			◐	●		●	●
Pea [25]	1986	◐	●	●	●		◐			●	●	●	●	●
Putnam et al. [26]	1986	●	●		●		◐				●	●		
Sleeman et al. [33]	1986	●	●		●		◐				●	◐		
Fleury [9]	1991	●	●		●		●				●			●
Holland et al. [13]	1997		●	◐	●					◐	●		◐	●
Madison and Gifford [22]	1997		●		●						●	●	●	◐
Fleury [10]	2000	●	●	●	●						●		◐	●
Hristova et al. [14]	2003		●		◐						●			●
Eckerdal and Thuné [8]	2005		●		●		◐			◐		●		●
Jackson et al. [15]	2005		●	●							●	◐		
Ragonis and Ben-Ari [28]	2005		●		●		◐				◐	◐		
Teif and Hazzan [40]	2006	●	●		●		◐			◐	●	●	●	
Doukakis et al. [6]	2007		●		●					◐	●		●	●
Ma [20]	2007	●	●	●	●						●	●		
Sorva [35]	2007		●		●				◐		●	●		●
Sajaniemi et al. [30]	2008	●	●	●	●						●	●	●	
Sorva [36]	2008		●		●				◐		●	●		●
Kaczmarczyk et al. [16]	2010	●	●		●		◐				●	●		●
Sirkia and Sorva [32]	2012	●	●	●	●		●		●		●			
Altadmri et al. [1]	2015	●	●				◐				●	◐		
Swidan et al. [39]	2018	●	●	●	●						●	●	◐	
Caceffo et al. [3]	2019	●	●		●		◐			●	●	◐		●

We are building

progmiscon.org

for

Professors • Teachers • TAs

CSEd Researchers

progmiscon.org

Based on
Luca Chiodini, Igor Moreno Santos, Andrea Gallidabino, Anya Tafliovich, André L. Santos, Matthias Hauswirth
A Curated Inventory of Programming Language Misconceptions
ITiCSE '21