From tests to proofs II

Why do we trust programs?

CS-214 - 08 Oct 2025

Clément Pit-Claudel

Quick announcements

Unguided callback ideas & rules are up!

Retour indicatif is now available! More polls!

Last week

Debugging II
Tests

Debugging “in the small”

Demo: `instrumentation`

Testing to avoid bugs

- Acceptance tests
- System tests
- Integration tests
- Unit tests
- Monitoring

Recap: Testing a theater play

Let's assume you're preparing a theater play.

1. You rehearse your monologue in front of the mirror

   ⇒ Unit test
2. You rehearse a scene in a classroom with the other actor

   ⇒ Integration test
3. You complete a full rehearsal in the actual theater

   ⇒ System test
4. You perform at the premiere

   ⇒ Acceptance test
5. The prompter backstage sees you struggle and feeds you line

   ⇒ Monitoring

Making trustworthy software II

Learning objectives:

1. Understand the behavior of complex recursive programs

2. Formulate software specifications

Specifications

From user stories to formal specs

Part I

An interactive debugging demo

Demo

Debugging a webapp

The 2024 CS214 guide to debugging: on one slide

Process

Triage phase

1. Check that there is a problem
2. Reproduce the issue
3. Decide whether it's your problem
4. Write it up

Diagnosis phase

1. Learn about the system
2. Observe the defect
3. Simplify and minimize to isolate
4. Guess and verify
5. Fix and confirm the fix
6. Prevent regressions

Techniques

- Keep notes
- Change one thing at a time
- Apply the scientific method
- Instrument
- Divide and conquer
- Ask for help

Pitfalls

- Random mutation
- Staring aimlessly
- Wasting time
- Assuming a bug went away
- Fixing effects, not causes
- Losing data

Part II

Specs: From English to Math

1. **User stories**
   Capture needs and goals of users
2. **Requirements**
   Say what the user wants
3. **Specifications**
   Say what the program does
4. **Formal specifications**
   Tie it all together with code & math

User stories: Capture what users care about

- **Use a few words to capture the essence of the project**

- **User, topic, purpose** structure
  → *Who, Where, What*

→ See *The Software Entreprise* in BA5

Examples:

- "As a **business analyst**, when visiting **the online dashboard**, I *want to be able to* **retrieve aggregate visitor statistics** over the last day, week, and month"
- "As a passenger, when opening the CFF timetable, I *want to* quickly find out which platform my train departs from"
- "By the end of the testing class, students *should be able to* identify, name, and describe the fives most important kinds of tests"
- "As a busy home cook, when preparing dinner, I *want to be able to* cook chickpeas in less than 15 minutes."

User stories: Capture what users care about

- **Use a few words to capture the essence of the project**

- **User, topic, purpose** structure
  → *Who, Where, What*

→ See *The Software Entreprise* in BA5

Examples:

- "As a **business analyst**, when visiting **the online dashboard**, I *want to be able to* **retrieve aggregate visitor statistics** over the last day, week, and month"
- "As a **passenger**, when opening **the CFF timetable**, I *want to* quickly **find out which platform** my train departs from"
- "By the end of **the testing class**, **students** *should be able to* **identify, name, and describe** the fives most important kinds of tests"
- "As a **busy home cook**, when **preparing dinner**, I *want to be able to* **cook chickpeas** in less than 15 minutes."

Requirements: Complete description

Functional reqs: concrete, testable objectives

→ *"The* `find` *function must return a boolean indicating whether results were found"*
→ *"The* `simplify` *function should be idempotent"*

Non-functional reqs: general properties

→ *"Each lab should be fun or useful"*
→ *"The API should respond quickly"*

Examples

- Project proposals
- Statements of work
- Improvement proposals (PIPs, SIPs, JEPs)
- Product requirement documents
- Intro of lab writeups!

Exercise: What kinds of requirements do our labs have?

JEP 459: String Templates (Second Preview)

Owner	Jim Laskey
Type	Feature
Scope	SE
Status	Candidate
Component	specification / language
Discussion	amber dash dev at openjdk dot org
Effort	M
Duration	M
Reviewed by	Gavin Bierman
Created	2023/08/14 12:17
Updated	2023/10/05 19:13
Issue	8314219

Summary

Enhance the Java programming language with *string templates*. String templates complement Java's existing string literals and text blocks by coupling literal text with embedded expressions and *template processors* to produce specialized results. This is a preview language feature and API.

History

String Templates were originally proposed as a preview feature by JEP 430 in JDK 21. We here propose another round of preview in order to gain additional experience and feedback. We do not propose any changes to the design.

Goals

- Simplify the writing of Java programs by making it easy to express strings that include values computed at run time.

https://openjdk.org/jeps/459

SIP-44 - FEWER BRACES

Search in doc...

By: Martin Odersky

History

Date	Version
July 1st 2022	Initial Draft
July 21st 2022	Expanded Other Conerns Section

Summary

The current state of Scala 3 makes braces optional around blocks and template definitions (i.e. bodies of classes, objects, traits, enums, or givens). This SIP proposes to allow optional braces also for function arguments. The advantages of doing so is that the language feels more systematic, and programs become typographically cleaner. The changes have been implemented and and made available under the language import `language.experimental.fewerBraces`. The proposal here is to make them available without a language import instead.

Motivation

After extensive experience with the current indentation rules I conclude that they are overall a big success. However, they still feel incomplete and a bit unsystematic since we can replace `{...}` in the majority of situations, but there are also important classes of situations where braces remain mandatory. In particular, braces are currently needed around blocks as function arguments.

It seems very natural to generalize the current class syntax indentation syntax to function arguments. In both cases, an indentation block is started by a colon at the end of a line. Doing so will bring two major benefits:

- Better *consistency*, since we avoid the situation where braces are sometimes optional and in other places mandatory.
- Better *readability* in many common use cases, similar to why current optional braces lead to better readability.

Proposed solution

Status

This proposal has been implemented, it will be available in the next minor release of the compiler.

SIP Contents

Problem with this page?
Please help us fix it!

https://docs.scala-lang.org/sips/fewer-braces.html

* ■ *Implement a small version of the `find` Unix command.*
  - Tell students to start early.
  * Students provide an implementation of 4 functions
    - Find all
    - Find files by name (exact)
    - Find large files (above a certain threshold)
    - Find empty files (size == 0)

    Each filter is a function that recurses down the file system and ***prints*** results (no collecting results in lists)
  * Bonus question (no points)

    Write a variant of each function that returns just the first result
  * File system entries are records with 6 fields:
    - IsDir?
    - Name
    - Full path
    - HasNext?
    - Pointer to the next entry at the same level (sibling, only valid if HasNext?)
    - Pointer to the first child entry (leftmost descendant, only valid for directories, may be Nil)
    - Size (only valid for files)
  * We expect students to write different functions for each predicate for now:
    - No higher order functions
    - No laziness/streaming
    - No async IO
    - Lots of code duplication is expected, reassure the students that we'll learn how to do better later.
  * We provide a small CLI utility that they can actually run on their computers.
  * For testing

    We link student code against a virtual FS (just a tree really).

Specifications: Unambiguous functional reqs

- Only covers functional requirements
- Can include functionality, performance, error handling, availability, …
- **Unambiguous**

Examples:

- IETF RFCs
- Language specs
- Implementation section of lab write-ups!

Requirements are for **customers**.

Specs are for **engineers**.

Interlude: Anyone speaks German here?

3.7.4 Zugbildung

Um das ungewollte Freimelden von Streckenabschnitten durch das Rückstellen der Achszähler auf Null und dadurch Zugsgefährdungen zu vermeiden, darf die effektive Gesamtachszahl eines Zuges nicht 256 Achsen betragen.

Regelwerk I-30111

SBB CFF FFS
bls
SOB SÜDOSTBAHN

Regelwerkversion	**6-0**	Vertraulichkeitsklassifikation	**intern**
gültig ab	**01.07.2013**	Eigner	**I-B-SBE-SNV**
letzte Review	**18.01.2013**	Betroffene Prozesse	----
nächste Review		verfügbare Sprachen	**DE, FR, IT**

Betroffene Divisionen	**Infrastruktur, Personenverkehr, Cargo**
Spezifische Empfänger / Verteiler	**EVU Lokführer und Leitung / LIDI: R I-30111**
Ersatz für	Version 5-0 sowie aufgehobene Vorschriften in Ziffer 6

Zuordnung:	R 300.1-15	Hauptgruppe I-B:	**R**

Interlude: Anyone speaks German here?

3.7.4 Formation des trains

In order to avoid falsely signalling a section of track as clear by resetting the axle counters to zero, and thus to avoid endangering trains, the effective total number of axles of a train must not equal 256 axles.

Regelwerk	I-30111	SBB CFF FFS / bls / SOB SÜDOSTBAHN

Regelwerkversion	**6-0**	Vertraulichkeitsklassifikation	**intern**
gültig ab	**01.07.2013**	Eigner	**I-B-SBE-SNV**
letzte Review	**18.01.2013**	Betroffene Prozesse	----
nächste Review		verfügbare Sprachen	**DE, FR, IT**

Betroffene Divisionen	**Infrastruktur, Personenverkehr, Cargo**
Spezifische Empfänger / Verteiler	**EVU Lokführer und Leitung / LIDI: R I-30111**
Ersatz für	Version 5-0 sowie aufgehobene Vorschriften in Ziffer 6

Zuordnung:	R 300.1-15	Hauptgruppe I-B:	**R**

Boids UI (partial) requirement:
“A UI will display the evolutions of boids over time at min 30FPS”.

Boid UI (partial) spec, whole application:
“An HTML canvas will be updated by a Javascript `requestAnimationFrame` loop that communicates with the server through `fetch`() calls to a REST api.”

aastore

Operation

Store into `reference` array

Format

```
aastore
```

Forms

aastore = 83 (0x53)

Operand Stack

..., *arrayref*, *index*, *value* →

...

Description

The *arrayref* must be of type `reference` and must refer to an array whose components are of type `reference`. The *index* must be of type `int` and value must be of type `reference`. The *arrayref*, *index*, and *value* are popped from the operand stack. The `reference` *value* is stored as the component of the array at *index*.

At run time, the type of *value* must be compatible with the type of the components of the array referenced by *arrayref*. Specifically, assignment of a value of reference type S (source) to an array component of reference type T (target) is allowed only if:

- If S is a class type, then:

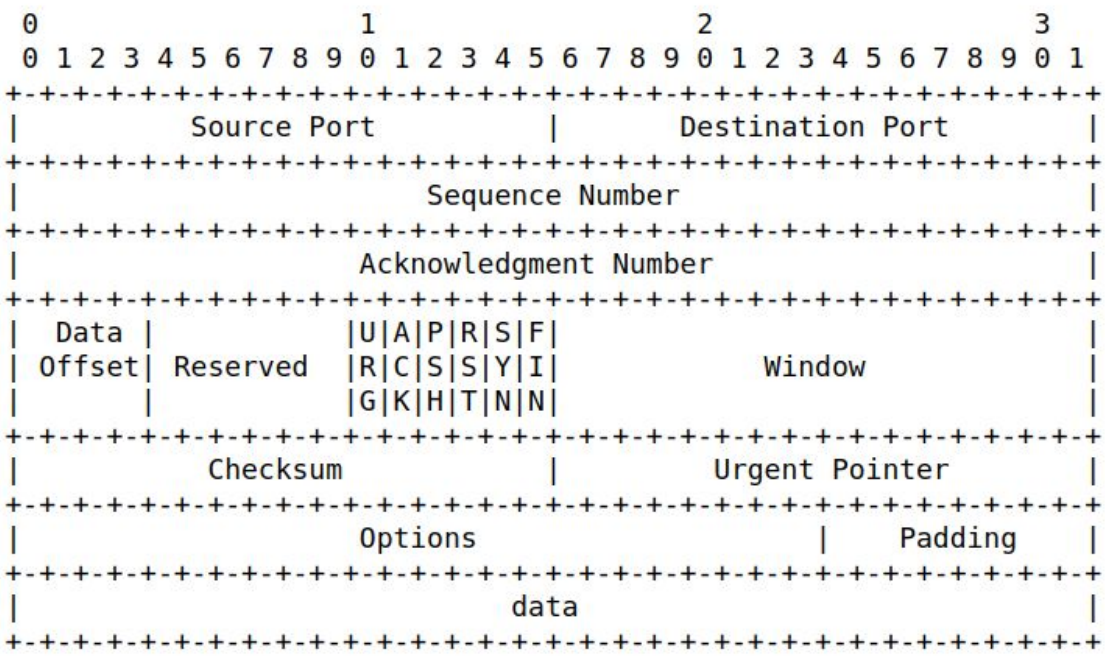
https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-6.html

3. FUNCTIONAL SPECIFICATION

3.1. Header Format

TCP segments are sent as internet datagrams. The Internet Protocol header carries several information fields, including the source and destination host addresses [2]. A TCP header follows the internet header, supplying information specific to the TCP protocol. This division allows for the existence of host level protocols other than TCP.

TCP Header Format



TCP Header Format

https://www.ietf.org/rfc/rfc793.txt

Implementation

You have five functions to implement: `findAllAndPrint`, `findByNameAndPrint`, `findBySizeEqAndPrint`, `findBySizeGeAndPrint`, `findEmptyAndPrint`. Each of them implements a different `find` filter (no filter, `-name`, `-size`, `-size +`, and `-empty`) by printing matching files and directories and returning a Boolean indicating whether any results were found. More precisely:

1. Each function takes one *entry* (a starting point for the search).
2. Some functions additionally take a *filter* (`name`, `size`, or `minSize`).
3. Each function must recursively check all files that are reachable from the initial *entry* parameter.
4. Each function should print the path (`.path()`) of any entry that matches the search criterion, and return a boolean indicating whether any matching entries were found.

Exercise:

Jass or Belote?

5.2. Règles

1. On doit toujours fournir la couleur demandée à l'entame si l'on en possède

2.1. Si l'on ne possède pas de carte dans la couleur demandée, et que notre partenaire est maître c'est-à-dire qu'il a joué la meilleure carte sur le tapis (on dit qu'il « tient le pli »): on peut alors jouer n'importe quelle carte; on se "défausse". On peut également jouer atout si bon nous semble.

2.2. Si l'on ne possède pas de carte dans la couleur demandée, et que notre partenaire n'est pas maître ou n'a pas encore joué: on est tenu de jouer un atout si l'on en possède (on dit que l'on « coupe »), sinon on se défausse en jouant n'importe quelle carte.
Précision: Si un adversaire a déjà coupé, et qu'il ne nous reste que des atouts inférieurs au sien, il n'est **pas** obligatoire d'en jouer un (on dit que l'on « ne pisse pas »), on peut se défausser.

3. Lorsque l'on est conduit à jouer atout, soit parce qu'il s'agit de la couleur demandée à l'entame, soit parce que l'on doit couper, on doit toujours jouer un atout plus fort que ceux déjà présents sur le tapis, même si cela est contraire à notre intérêt. Si cela s'avère impossible, on peut jouer un atout plus faible.

4. Lorsque notre partenaire, maître, a coupé une carte adverse et que nous ne possédons plus que de l'atout, il n'est pas obligé de fournir un atout supérieur. C'est le seul cas de figure, plutôt rare, où il est permis de jouer un atout inférieur.

https://www.ffbelote.org/wp-content/uploads/2015/11/REGLES-DE-LA-BELOTE-COINCHEE.pdf

Jeu de la carte

Le joueur qui a la main pour le pli joue une carte de son choix. Il peut jouer un atout ou une carte d'une autre couleur quelconque. La couleur de la carte jouée détermine alors la couleur demandée pour le pli. Le joueur suivant dans le sens inverse des aiguilles d'une montre joue alors une carte de son choix en respectant les règles suivantes. Si la couleur demandée est la couleur d'atout, il doit fournir un atout s'il peut et sinon il défausse une carte de son choix. Si la couleur demandée n'est pas la couleur d'atout, il peut « couper » d'un atout supérieur aux atouts déjà joués lors de ce pli. S'il décide de ne pas couper, il doit fournir la couleur demandée, mais peut défausser une carte de son choix s'il ne peut pas fournir.

Lorsque l'on n'a plus que des atouts en main, il est permis de sous-couper en jouant un atout plus faible que les atouts déjà joués. Le valet d'atout peut être joué à n'importe quel moment du jeu, on n'est pas obligé de le fournir à l'atout.

Lorsque chacun des quatre joueurs a joué une carte, le pli est terminé. Le pli est remporté par le joueur ayant joué l'atout le plus élevé ou la carte la plus élevée de la couleur demandée si aucun atout n'est joué. L'équipe de ce joueur remporte les quatre cartes jouées sur ce pli. Le joueur remportant le pli gagne la main pour le pli suivant.

https://coinche.ch/regles.pdf

Problem

English and French are horrible languages for specifications

A small sample of ambiguities

Attachment issues

"Each subscriber to a newspaper based in Morges, VD will be contacted."

Logical ambiguity

"The system shall permit access to employees who work in Building D and Building E."

Word ambiguity

"*Time flies like an arrow, fruit flies like a banana"*

Numbers

"Access will be restricted to users more than 13 years old"

Nested negations

"It is not as if there aren't technologies at hand today that wouldn't improve the toasting experience if thoughtfully incorporated into a new generation of toasting devices."

Jonah Goldberg, "Whither the Toaster?", NRO 9/10/2012

etc., etc.

Exercise: rephrase these as logical propositions

https://www.archives.gov/federal-register/write/legal-docs/ambiguity.html

A small sample of ambiguities

```
val contact = journals.exist.(user.subd) && user.inMorges
val contact = journals.filter(_.inMorges).exists(user.subd)
```

"Each subscriber to a newspaper based in Morges, VD will be contacted."

```
val access = employee.worksIn(B) && employee.worksIn(D)
val access = employee.worksIn(B) || employee.worksIn(D)
```

"The system shall permit access to employees who work in Building D and Building E."

```
time.fliesLike(arrow)
fruitFlies.enjoy(banana)
```

"Time flies like an arrow, fruit flies like a banana"

```
val access = user.age >  13.0
val access = user.age ⩾ 14
```

"Access will be restricted to users more than 13 years old"

```
!(!tech.exists(!_.improvesToasting))
```

equivalent to `tech.exists(!_.improvesToasting)` *(woops!)*

"It is not as if there aren't technologies at hand today that wouldn't improve the toasting experience if thoughtfully incorporated into a new generation of toasting devices."

Jonah Goldberg, "Whither the Toaster?", NRO 9/10/2012

etc., etc.

https://www.archives.gov/federal-register/write/legal-docs/ambiguity.html

Formal specs: tying it all together with math & code

- Like a spec, but **written in a restricted, unambiguous language**
  → Math or code
  → Supports *formal proofs*

Formal specs are for **engineers & computers**.

Examples:

- Software models
  → Reference implementation
- Axiomatic specs
  → Parametric integration tests
  → Monitors

Two classes at EPFL:

- *Formal Verification*
  (V. Kunčak, IN)
- *Interactive theorem proving*
  (C. Pit-Claudel, EDIC)

Examples of formal specs

- **Software model**

```
fastExponentiation(x: Int, n: Int) …
ensuring (res ⇒
  res == slowExponentiation(x, n))

fastExponentiation(x: Double, n: Int) …
ensuring (res ⇒
  abs(res - slowExponentiation(x, n)) < ε)
```

Examples of formal specs

- **Axiomatic (monitor)**

```
filter[A](l: List[T], p: T ⇒ Boolean) …
ensuring (res ⇒
  res.forall(p) &&
  res.subsequence(l) &&
  l.forall(t ⇒ res.contains(t) || !p(t)))
```

- **Axiomatic (parametric integration test)**

```
def pushPop[T](s: Stack[T], t: T) =
  assert(pop(push(t, s)) == (t, s)
```

2.3.1. The ChaCha20 Block Function in Pseudocode

Note: This section and a few others contain pseudocode for the algorithm explained in a previous section. Every effort was made for the pseudocode to accurately reflect the algorithm as described in the preceding section. If a conflict is still present, the textual explanation and the test vectors are normative.

```
inner_block (state):
   Qround(state, 0, 4, 8,12)
   Qround(state, 1, 5, 9,13)
   Qround(state, 2, 6,10,14)
   Qround(state, 3, 7,11,15)
   Qround(state, 0, 5,10,15)
   Qround(state, 1, 6,11,12)
   Qround(state, 2, 7, 8,13)
   Qround(state, 3, 4, 9,14)
   end

chacha20_block(key, counter, nonce):
   state = constants | key | counter | nonce
   working_state = state
   for i=1 upto 10
      inner_block(working_state)
      end
   state += working_state
   return serialize(state)
   end
```

Specs at all levels

```
def groupBy[K](f: (A) => K): Map[K, List[A]]
```

Partitions this iterable collection into a map of iterable collections according to some discriminator function.

Note: Even when applied to a view or a lazy collection it will always force the elements.

K	the type of keys returned by the discriminator function.
f	the discriminator function.
returns	A map from keys to iterable collections such that the following invariant holds:

```
(xs groupBy f)(k) = xs filter (x => f(x) == k)
```

That is, every key k is bound to a iterable collection of those elements x for which `f(x)` equals k.

Formal specs: tying it all together with math & code

Exercise: Fix the following spec:

```
def maxIncomplete(l: List[Int]): Int =
{
  requires(!l.isEmpty)
  …
} ensuring (res ⇒ l.forall(x ⇒ x ⩽
res))
```

Formal specs: tying it all together with math & code

Exercise: Fix the following spec:

```
def maxIncomplete(l: List[Int]): Int = {
  requires(!l.isEmpty)
  …
} ensuring (res ⇒ l.forall(x ⇒ x ⩽ res))
```

Solution:

```
def maxComplete(l: List[Int]): Int = {
  requires(!l.isEmpty)
  …
} ensuring (res ⇒ l.forall(x ⇒ x ⩽ res)
                    && l.contains(res))
```

Formal specs: tying it all together with math & code

Exercise: Fix the following spec:

```
def sort(l: List[Int]): List[Int] = {
  …
} ensuring (res ⇒
    (0 to l.length - 2).forall(idx ⇒
      res(idx) ⩽ res(idx + 1)))
```

Formal specs: tying it all together with math & code

Exercise: Fix the following spec:

```
def sort(l: List[Int]): List[Int] = {
  …
} ensuring (res ⇒
    (0 to l.length - 2).forall(idx ⇒
      res(idx) ⩽ res(idx + 1)))
```

Solution:

```
def sort(l: List[Int]): List[Int] = {
  …
} ensuring (res ⇒
    l.length == res.length &&
    (0 to l.length - 2).forall(idx ⇒
      res(idx) ⩽ res(idx + 1)) &&
    l.forall(x ⇒ res.count(_ == x) == l.count(_ == x)))
```

Recap: specs and tests

- **User stories** capture the *who* (target audience) and *what* (needs and goals)
- **Requirements** (functional / non-functional) are acceptance criteria
- **Specifications** restate requirements in detailed language
- **Formal specifications** capture requirements unambiguously for all inputs

- **Acceptance and system tests** validate against requirements
- **Integration tests, unit tests, and monitors** validate against specifications

If we're early: fill in the

***bref retour indicatif* !**