Java Expressions & Classes

M250 Tutorial 02

Phil Molyneux

9 November 2025

Java Expressions & Classes

M250 Tutorial Agenda

- Introductions
- Adobe Connect reminders
- *Adobe Connect* — if you or I get cut off, wait till we reconnect (or send you an email)
- Types
- Data types & variables
- Expressions
- Classes
- Some useful Web & other references
- Time: about 1 hour
- Do ask questions or raise points.
- Slides/Notes M250Tutorial20251109ExpressionsPrsntn2025J

M250 Tutorial

Introductions — Phil

- *Name* Phil Molyneux
- *Background*
    - Undergraduate: Physics and Maths (Sussex)
    - Postgraduate: Physics (Sussex), Operational Research (Brunel), Computer Science (University College, London)
    - Worked in Operational Research, Business IT, Web technologies, Functional Programming
- *First programming languages* Fortran, BASIC, Pascal
- *Favourite Software*
    - Haskell — pure functional programming language
    - Text editors TextMate, Sublime Text — previously Emacs
    - Word processing in LaTeX — all these slides and notes
    - Mac OS X
- *Learning style* — I read the manual before using the software

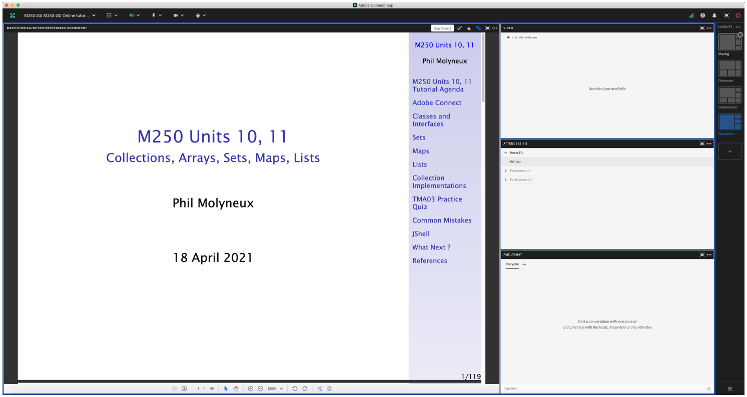
M250 Tutorial

Introductions — You

- *Name* ?
- *Favourite software/Programming language* ?
- *Favourite text editor or integrated development environment (IDE)*
- List of text editors, Comparison of text editors and Comparison of integrated development environments
- *Other OU courses* ?
- *Anything else* ?

Adobe Connect

Interface — Host View



M250 Units 10, 11

Collections, Arrays, Sets, Maps, Lists

Phil Molyneux

18 April 2021

M250 Units 10, 11

Phil Molyneux

M250 Units 10, 11 Tutorial Agenda
Adobe Connect
Classes and Interfaces
Sets
Maps
Lists
Collection Implementations
TMA03 Practice Quiz
Common Mistakes
JShell
What Next ?
References

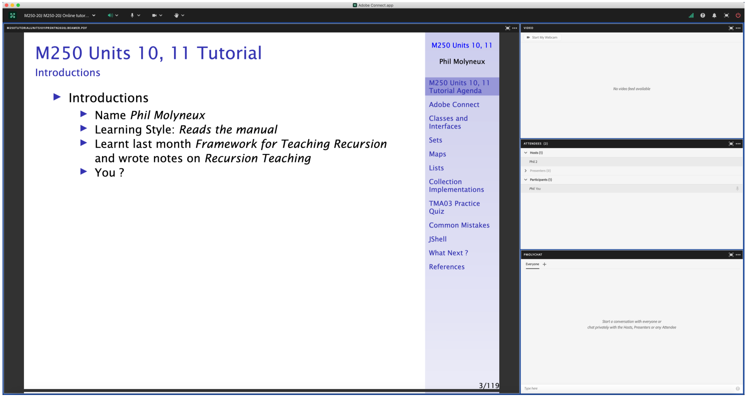
1/119

No video feed available

Start a conversation with everyone or
chat privately with the Hosts, Presenters or any Attendee

Adobe Connect

Interface — Participant View


Adobe Connect.app

M250-20J M250-20J Online tutor...

M250TUTORIALUNITS1011PRSNTN2020J.BEAMER.PDF

M250 Units 10, 11 Tutorial

Introductions

- Introductions
  - Name *Phil Molyneux*
  - Learning Style: *Reads the manual*
  - Learnt last month *Framework for Teaching Recursion* and wrote notes on *Recursion Teaching*
  - You ?

M250 Units 10, 11

Phil Molyneux

M250 Units 10, 11 Tutorial Agenda
Adobe Connect
Classes and Interfaces
Sets
Maps
Lists
Collection Implementations
TMA03 Practice Quiz
Common Mistakes
JShell
What Next ?
References

3/119

VIDEO

Start My Webcam

No video feed available

ATTENDEES (2)

Hosts (1)
Phil 2
Presenters (0)
Participants (1)
Phil You

PMOLYCHAT

Everyone

Start a conversation with everyone or chat privately with the Hosts, Presenters or any Attendee

Type here

Adobe Connect

Settings

- **Everybody** *Menu bar* › *Meeting* › Speaker & Microphone Setup
- *Menu bar* › *Microphone* › Allow Participants to Use Microphone ✓
- Check Participants see the entire slide **Workaround**
  - *Disable Draw* *Share pod* › *Menu bar* › *Draw icon*
  - *Fit Width* *Share pod* › *Bottom bar* › *Fit Width icon* ✓
- *Meeting* › Preferences › General › **Host Cursor** › Show to all attendees
- *Menu bar* › *Video* › Enable **Webcam** for Participants ✓
- Do not *Enable single speaker mode*
- Cancel hand tool
- Do not enable green pointer
- **Recording** Meeting › Record Session ✓
- **Documents** Upload PDF with drag and drop to share pod
- Delete *Meeting* › Manage Meeting Information › Uploaded Content and *check filename* › *click on delete*

Adobe Connect

Access

- **Tutor Access**

  TutorHome › M269 Website › Tutorials

  Cluster Tutorials › M269 Online tutorial room

  Tutor Groups › M269 Online tutor group room

  Module-wide Tutorials › M269 Online module-wide room
- **Attendance**

  TutorHome › Students › View your tutorial timetables
- **Beamer Slide Scaling** 440% (422 x 563 mm)
- **Clear Everyone's Status**

  Attendee Pod › Menu › Clear Everyone's Status
- **Grant Access** and send link via email

  Meeting › Manage Access & Entry › Invite Participants. . .
- **Presenter Only Area**

  Meeting › Enable/Disable Presenter Only Area

Adobe Connect

Keystroke Shortcuts

- Keyboard shortcuts in Adobe Connect
- **Toggle Mic** ⌘+M (Mac), Ctrl+M (Win) (On/Disconnect)
- **Toggle Raise-Hand status** ⌘+E
- **Close dialog box** ⎋ (Mac), Esc (Win)
- **End meeting** ⌘+\

Adobe Connect Interface

Sharing Screen & Applications

- Share My Screen ❯ Application tab ❯ Terminal for Terminal
- Share menu ❯ Change View ❯ Zoom in for mismatch of screen size/resolution (Participants)
- (Presenter) Change to 75% and back to 100% (solves participants with smaller screen image overlap)
- Leave the application on the original display
- Beware blued hatched rectangles — from other (hidden) windows or contextual menus
- Presenter screen pointer affects viewer display — beware of moving the pointer away from the application
- First time: System Preferences ❯ Security & Privacy ❯ Privacy ❯ Accessibility

Adobe Connect
Ending a Meeting

- *Notes for the tutor only*
- **Student:** Meeting › Exit Adobe Connect
- **Tutor:**
- **Recording** Meeting › Stop Recording ✓
- **Remove Participants** Meeting › End Meeting. . . ✓
  - Dialog box allows for message with default message:
  - *The host has ended this meeting. Thank you for attending.*
- **Recording availability** *In course Web site for joining the room, click on the eye icon in the list of recordings under your recording* — edit description and name
- **Meeting Information** Meeting › Manage Meeting Information — can access a range of information in Web page.
- **Delete File Upload** Meeting › Manage Meeting Information › Uploaded Content tab select file(s) and click Delete
- **Attendance Report** see course Web site for joining room

Adobe Connect

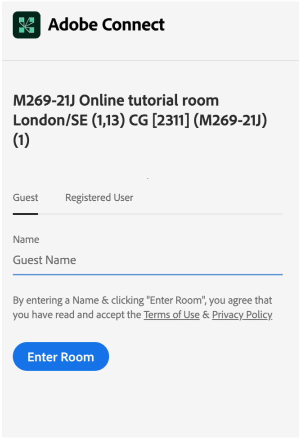
Invite Attendees

- **Provide Meeting URL** Menu › Meeting › Manage Access & Entry › Invite Participants. . .
- **Allow Access without Dialog** Menu › Meeting › Manage Meeting Information provides new browser window with *Meeting Information* Tab bar › Edit Information
- Check *Anyone who has the URL for the meeting can enter the room*
- Default *Only registered users and accepted guests may enter the room*
- **Reverts to default next session but URL is fixed**
- Guests have blue icon top, registered participants have yellow icon top — same icon if URL is open
- See Start, attend, and manage Adobe Connect meetings and sessions

Adobe Connect

Entering a Room as a Guest (1)

- Click on the link sent in email from the Host
- Get the following on a Web page
- As *Guest* enter your name and click on Enter Room


Adobe Connect

M269-21J Online tutorial room London/SE (1,13) CG [2311] (M269-21J) (1)

Guest Registered User

Name

Guest Name

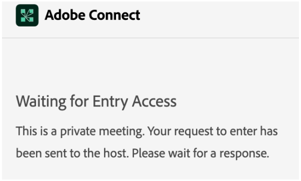
By entering a Name & clicking "Enter Room", you agree that you have read and accept the Terms of Use & Privacy Policy

Enter Room

Adobe Connect

Entering a Room as a Guest (2)

- See the *Waiting for Entry Access* for *Host* to give permission



Adobe Connect

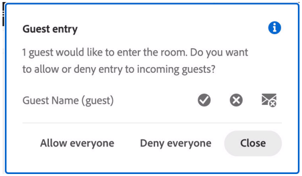
Waiting for Entry Access

This is a private meeting. Your request to enter has been sent to the host. Please wait for a response.

Adobe Connect

Entering a Room as a Guest (3)

- *Host* sees the following dialog in *Adobe Connect* and grants access



Guest entry

1 guest would like to enter the room. Do you want to allow or deny entry to incoming guests?

Guest Name (guest)

Adobe Connect

Layouts

- **Creating new layouts** example *Sharing* layout
- Menu › Layouts › Create New Layout. . . › Create a New Layout dialog › Create a new blank layout and name it *PMolyMain*
- New layout has no Pods but does have Layouts Bar open (see Layouts menu)
- **Pods**
- Menu › Pods › Share › Add New Share and resize/position — initial name is *Share n* — rename *PMolyShare*
- **Rename Pod** Menu › Pods › Manage Pods. . . › Manage Pods › Select › Rename or Double-click & rename
- Add Video pod and resize/reposition
- Add Attendance pod and resize/reposition
- Add Chat pod — rename it *PMolyChat* — and resize/reposition

Adobe Connect

Layouts

- Dimensions of **Sharing** layout (on 27-inch iMac)
  - Width of Video, Attendees, Chat column 14 cm
  - Height of Video pod 9 cm
  - Height of Attendees pod 12 cm
  - Height of Chat pod 8 cm
- **Duplicating Layouts** does *not* give new instances of the Pods and is probably not a good idea (apart from local use to avoid delay in reloading Pods)
- **Auxiliary Layouts** name *PMolyAux0n*
  - Create new Share pod
  - Use existing Chat pod
  - Use same Video and Attendance pods

Adobe Connect

Chat Pods

- **Format Chat text**
- Chat Pod › menu icon › My Chat Color
- Choices: Red, Orange, Green, Brown, Purple, Pink, Blue, Black
- Note: Color reverts to Black if you switch layouts
- Chat Pod › menu icon › Show Timestamps

Graphics Conversion

PDF to PNG/JPG

- Conversion of the screen snaps for the installation of Anaconda on 1 May 2020
- Using GraphicConverter 11
- File › Convert & Modify › Conversion › Convert
- Select files to convert and destination folder
- Click on Start selected Function or ⌘+⏎

Adobe Connect Recordings

Exporting Recordings

- *Menu bar* › *Meeting* › Preferences › Video
- Aspect ratio › Standard (4:3) (not Wide screen (16:9) default)
- Video quality › Full HD (1080p not High default 480p)
- **Recording** *Menu bar* › *Meeting* › Record Session ✓
- **Export Recording**
- *Menu bar* › *Meeting* › Manage Meeting Information
- *New window* › Recordings › *check Tutorial* › Access Type button
- check Public › check Allow viewers to download
- **Download Recording**
- *New window* › Recordings › *check Tutorial* › Actions › Download File

Types

Types & Type Systems (1)

- Types are collections of related values and operations on them
- Different programming languages take different views about what primitive and structured types are available and the extent to which users can define their own types
- Type systems are syntactic methods for assigning a type to each expression in the programming language
- Type systems address the problem of ensuring that programs have meaning

Types

Types & Type Systems (2)

- The aim of a Type System is to prove the absence of certain program behaviours by answering the following:
- **Type checking** — given the type declaration for variables, methods or constructors are they used consistently in expressions — in Java this is called *static* since it is checked at compile time
- **Type compatibility and equivalence** — this can be *nominal* based on declarations and names, or *structural* based on the characteristics of the types — in Java this is *nominal*
- **Type inference** — given an expression, what is its most general type? In Java it requires explicit declarations

Types

Types & Type Systems (3)

- Type safety — the only operations that can be performed on data are those permitted by the type of the data
- Polymorphism provides a single interface to entities of different types
- Ad hoc polymorphism also known as *function or operator overloading*
- Parametric polymorphism — called *Generics* in Java

Types

Types in Java

- A *type* is a set of values and operations on them
- A type is either a *primitive* type or a *reference* type
- **Primitive Type** is either `boolean` or one of the *numeric* types `char`, `byte`, `short`, `int`, `long`, `float`, `double`
- The integer types use signed two's complement representation (except for `char`)
- If the most positive is *max* then the most negative is $-max - 1$
- The floating-point types follow the IEEE 754 standard
- M250 restricts itself to `int` and `double` — see Unit 3 Appendix 1

Types in Java

Reference Types

- A **Reference Type** is a class type defined by a class declaration or an interface type defined by an interface declaration or an array type or an enum type
- An **array** is an indexed collection of variables, called elements
- Arrays are covered in Unit 9 in M250
- An *enum* type is used to declare enumerated values which have order — for example, days of the week or months for dates
- Enum types are not covered in M250 — enumeration types are special case of Algebraic Data Types
- A value of reference type is either `null` or a reference to an object or array
- The special value `null` denotes *no object*
- The literal `null`, denoting the `null` value, can have any reference type

Types

Primitive Types

Type	Literals	Range	Wrapper
boolean	false, true		Boolean
char	' ', 'a', ...	\u0000 ... \uFFFF Unicode	Character
byte	0, 1, ...	$max = 127 = 2^7 - 1$	Byte
short	0, 1, ...	$max = 2^{15} - 1$ 32767	Short
int	0, 1, ...	$max = 2^{31} - 1$ 2147483647	Integer
long	0L, 1L, ...	$max = 2^{63} - 1$ 9223372036854775807	Long
float	0.49f, 3E8f, ...	$\pm 10^{-38} \cdots \pm 10^{38}$	Float
double	0.49, 3E8, ...	$\pm 10^{-308} \cdots \pm 10^{308}$	Double

Types

Type Conversion

```
/* (a) */ 1 + 2
/* (b) */ 1 + '2'
/* (c) */ 1 + "2"
/* (d) */ 3 * 5
/* (e) */ 3 * '5'
/* (f) */ 1 + 2 + 3
/* (g) */ 1 + 2 + '3'
/* (h) */ 1 + 2 + "3"
/* (i) */ '1' + 2 + 3
/* (i) */ "1" + 2 + 3
```

Types

Type Conversion

```
/* (a) */ 1 + 2         == 3
/* (b) */ 1 + '2'       == 51
  jshell> int x = (int) '2'
  x ==> 50
/* (c) */ 1 + "2"
  jshell> String s = 1 + "2"
  s ==> "12"
/* (d) */ 3 * 5         == 15
/* (e) */ 3 * '5'
  jshell> int y = 3 * '5'
  y ==> 159
```

- `(int) '2'` is a *type cast expression*

Types

Type Conversion

```
/* (f) */ 1 + 2 + 3     == 6
/* (g) */ 1 + 2 + '3'
  jshell> int z = 1 + 2 + '3'
  z ==> 54
/* (h) */ 1 + 2 + "3"
  jshell> String t = 1 + 2 + "3"
  t ==> "33"
/* (i) */ '1' + 2 + 3
  jshell> int a = '1' + 2 + 3
  a ==> 54
/* (j) */ "1" + 2 + 3
  jshell> String w = "1" + 2 + 3
  w ==> "123"
```

```
jshell> String cs = "a"
cs ==> "a"

jshell> int codePoint = Character.codePointAt(cs,0)
codePoint ==> 97
```

Types

Type Conversion — Numbers to Strings

```
jshell> int x = 42
x ==> 42

jshell> String s1 = x + ""
s1 ==> "42"

jshell> String s2 = String.valueOf(x)
s2 ==> "42"

jshell> String s3 = Integer.toString(x)
s3 ==> "42"
```

Types

Type Conversion — Strings to Numbers

```
jshell> String s = "42"
s ==> "42"

jshell> Integer x = Integer.valueOf(s)
x ==> 42

jshell> int y = Integer.valueOf(s)
y ==> 42
```

Types

Type Conversion — char to String (1)

- From StackOverflow: Convert char to String

```
jshell> String strValOf =                           // (1)
   ...>   String.valueOf('c');      // most efficient
strValueOf ==> "c"

jshell> String strValOfCharArray =                  // (2)
   ...>   String.valueOf(new char[] {'c'});
strValOfCharArray ==> "c"

jshell> String charToStr =                          // (3)
   ...>   Character.toString('c');
charToStr ==> "c"
```

continued on next slide

Types

Type Conversion — char to String (2)

```
jshell> String charObjToStr =
   ...>   new Character('c').toString()       // (4)
charObjToStr ==> "c"

jshell> String concatBlankStr =               // (5)
   ...>   'c' + ""
concatBlankStr ==> "c"

// above looks simple but less efficient since
// concatenation expands to
// new StringBuilder().append(x).append("").toString()

jshell> String fromCharArray =                // (6)
   ...>   new String(new char[] {'c'})
fromCharArray ==> "c"
```

Data Types

Strings

- A *string* is an object of the built-in class `String`
- A `String` object is *immutable*
- A string *literal* is a sequence of characters in double quotes
- A string is stored as a 16 bit Unicode encoding — the first 128 characters are the ASCII character encoding

Strings

Escape Sequences for Character and String Literals

Escape code	Meaning	Unicode
\b	backspace BS	\u0008
\t	horizontal tab HT	\u0009
\n	linefeed LF	\u000A
\f	formfeed FF	\u000C
\r	carriage return CR	\u000D
\"	double quote "	\u0022
\'	single quote '	\u0027
\\\\	backslash \\	\u005C
\ddd	octal escape	\u0000 to \u00FF
\u*dddd*	Unicode escape	\u0000 to \uFFFF

- **Line Terminator**
  - ASCII linefeed LF, also known as *newline* *(Unix, macOS)*
  - ASCII carriage return CR, also known as *return* *(Microsoft Windows, MS-DOS)*
  - CR followed by LF *(Classic Mac OS)*
- Because Unicode escapes are processed early, use \n, \r and not the Unicode escapes (otherwise the Java processor will see a Line Terminator unescaped)

Strings
String Methods (1)

Below `s1` and `s2` are expressions of type `String` and `v` is an expression of any type

- `int s1.length()` returns the number of characters
- `boolean s1.equals(s2)` returns `true` if `s1` and `s2` have the same sequence of characters
- `char s1.charAt(i)` returns the character at position `i` in `s1` counting from `0`
- `String s1.toString()` is the same object as `s1`
- `String.valueOf(v)` returns the string representation of `v`, which can have any primitive or reference type.

  When `v` is not `null` then it is converted with `v.toString()`

  Any class `C` inherits from `Object` a default `toString()` method that produces strings of the form `C@2a5734` where `2a5734` is some memory address, but `toString()` may be overridden

Strings

String Methods (2)

- `s1 + s2` returns a new string which is the concatenation of `s1` and `s2`, it is the same as `String s1.concat(s2)`

  Both `s1 + v` and `v + s1` are evaluated by converting `v` to a string with `String.valueOf(v)`, thus using `v.toString()` when `v` has reference type
- See Exam Handbook and String documentation

Strings

String Methods (3)

- `int s1.compareTo(s2)`
- See Exam Handbook and String documentation

```
jshell> String s1 = "abc"
s1 ==> "abc"

jshell> String s2 = "ABC"
s2 ==> "ABC"

jshell> int compInt = s1.compareTo(s2)
compInt ==> 32

jshell> int s1Chr0 = s1.charAt(0)
s1Chr0 ==> 97

jshell> int s2Chr0 = s2.charAt(0)
s2Chr0 ==> 65
```

Strings

String Exercises (1)

```java
String s1 = "abc";
String s2 = s1 + "";
String s3 = s1;
String s4 = s1.toString()

/* (a) */ boolean b1 = s1 == s2;
/* (b) */ boolean b2 = s1 == s3;
/* (c) */ boolean b3 = s1 == s4;
/* (d) */ boolean b4 = s1.equals(s2);
/* (e) */ boolean b5 = s1.equals(s3);
```

Strings

String Exercises (2)

```
jshell> String s1 = "abc"
s1 ==> "abc"

jshell> String s2 = s1 + ""
s2 ==> "abc"

jshell> /* (a) */ boolean b1 = s1 == s2
b1 ==> false
```

- Note that (==) tests objects for *identity* not *equality*
- Two values of primitive type are equal if they represent the same value (after conversion to their common supertype)
- For example 10 and 10.0 are equal

Strings

String Exercises (3)

```
jshell> String s1 = "abc"
s1 ==> "abc"

jshell> String s3 = s1
s3 ==> "abc"

jshell> /* (b) */ boolean b2 = s1 == s3
b2 ==> true
```

- Note that (==) tests objects for *identity* not *equality*
- Two values of reference type are equal if they are both `null` or both references to the same object or array, created by the same boxing operation or execution of the `new` operator

Strings

String Exercises (4)

```
jshell> String s1 = "abc"
s1 ==> "abc"

jshell> String s4 = s1.toString()
s4 ==> "abc"

jshell> /* (c) */ boolean b3 = s1 == s4
b3 ==> true
```

Strings

String Exercises (5)

```
jshell> String s1 = "abc"
s1 ==> "abc"

jshell> String s2 = s1 + ""
s2 ==> "abc"

jshell> /* (d) */ boolean b4 = s1.equals(s2)
b4 ==> true
```

Strings

String Exercises (6)

```
jshell> String s1 = "abc"
s1 ==> "abc"

jshell> String s3 = s1
s3 ==> "abc"

jshell> /* (e) */ boolean b5 = s1.equals(s3)
b5 ==> true
```

Strings

String Exercises (7) — Equality

```
String sa = "abcd";
String sb = "abcd";
String sc = new String("abcd");
String sd = new String("abcd");

/* (a) */ boolean bab = sa == sb;
/* (b) */ boolean bcd = sc == sd;
```

Strings

String Exercises (8) — Equality

```
jshell> String sa = "abcd"
sa ==> "abcd"

jshell> String sb = "abcd"
sb ==> "abcd"

jshell> boolean bab = sa == sb
bab ==> true

jshell> String sc = new String("abcd")
sc ==> "abcd"

jshell> String sd = new String("abcd")
sd ==> "abcd"

jshell> boolean bcd = sc == sd
bcd ==> false
```

- What is going on here ?

Strings

String Exercises (9) — Equality

- A pool of strings is maintained by the class `String`
- When the `intern()` method is invoked, if the pool contains a string equal to this `String` object as determined by `equals()` then the pool string is used
- `s1.intern() == s2.intern()` if and only if `s1.equals(s2)`
- All string literals are interned

```
jshell> String se = new String("abcd").intern()
se ==> "abcd"

jshell> String sf = new String("abcd").intern()
sf ==> "abcd"

jshell> boolean bef = se == sf
bef ==> true

jshell> boolean baebf = sa == se && sb == sf
baebf ==> true
```

Strings

String Exercises (10) — Equality

- M250 does not mention `intern()`
- Always use `s1.equals(s2)` not `s1 == s2`
- See Create Java String Using " " or Constructor?
- What is Java interning
- Examples are given in the Java Language Specification — section 3.10.5 *String Literals* in edition 13

Type Compatibility and Assignment

Compatibility of Class Types (1)

```
Frog kermit = null;
Frog gribbit = null;
HoverFrog happy = null;
HoverFrog bouncy = null;

/* (a) */ kermit = new Frog();
/* (b) */ happy = new HoverFrog();
/* (c) */ kermit.right();
/* (d) */ kermit.right();
/* (e) */ happy.setColour(OUColour.RED);
/* (f) */ gribbit = kermit;
/* (g) */ gribbit.setColour(OUColour.BLUE);
/* (h) */ kermit.right();
/* (i) */ bouncy = kermit;
/* (j) */ kermit = happy;
/* (k) */ happy.up();
/* (l) */ kermit.up();
```

- Describe the effect of each of the above lines of code (or if a line does not compile)
- From Unit 3 Section 4.2 **Compatibility of class types** *Activity 7*

Type Compatibility and Assignment

Compatibility of Class Types (2)

- Describe the effect of each of the above lines of code (or if a line does not compile)
- Lines 1 to 4 create `Frog` and `HoverFrog` reference variables initialised to the value `null`.

(a) Line 6 `kermit` references a `Frog` with `position` set to 1 and `colour` set to `OUColour.GREEN`.

(b) Line 7 `happy` references a `HoverFrog` with `position` set to 1, `colour` set to `OUColour.GREEN` and height set to 0.

(c) Line 8 The `Frog` object referenced by `kermit` now has `position` set to 2.

(d) Line 9 The `Frog` object referenced by `kermit` now has `position` set to 3.

Type Compatibility and Assignment

Compatibility of Class Types (3)

- Describe the effect of each of the above lines of code (or if a line does not compile)

(e) Line `10` The `HoverFrog` object referenced by `happy` now has `colour` set to `OUColour.RED`.

(f) Line `11` `gribbit` now references the *same* `Frog` object as `kermit`. That object has `position` set to 3 and `colour` set to `OUColour.GREEN`. What happened here was that the reference stored in `kermit` was *copied* into `gribbit`.

(g) Line `12` The `Frog` object referenced by both `kermit` and `gribbit` now has `colour` set to `OUColour.BLUE`.

(h) Line `13` The `Frog` object referenced by both `kermit` and `gribbit` now has `position` set to 4.

Type Compatibility and Assignment

Compatibility of Class Types (4)

- Describe the effect of each of the above lines of code (or if a line does not compile)

(i) Line 14 This code does not compile and so `bouncy` has not been assigned any value.
The reference `kermit` is of type `Frog` and you cannot assign a `Frog` type reference to a variable of type `HoverFrog`. More informally we say, you cannot assign a `Frog` object to a `HoverFrog` variable. You should have seen an error message that included the words:
`incompatible types - found Frog but expected HoverFrog`.

Type Compatibility and Assignment

Compatibility of Class Types (5)

- Describe the effect of each of the above lines of code (or if a line does not compile)

(j) Line 15 It turns out that you can assign a reference to a HoverFrog object to a variable of type Frog. So kermit = happy results in the reference in happy being copied into kermit. This results in both kermit and happy referencing the same HoverFrog. The HoverFrog object has position set to 1, colour set to OUColour.RED, and height set to 0.

Type Compatibility and Assignment

Compatibility of Class Types (6)

- Describe the effect of each of the above lines of code (or if a line does not compile)

(k) Line 16 The `HoverFrog` object referenced by both `happy` and `kermit` now has `height` set to 1.

(l) Line 17 Although the variable `kermit` now references an instance of `HoverFrog`, the variable was declared as being of type `Frog`. As `up()` is not in the protocol of `Frog`, the compiler rejects the statement with an error message including the words: `cannot find symbol - method up()`

Expressions

Expr	Meaning	A	Arg(s)	Result
`!e`	logical negation	**R**	boolean	boolean
`e1 * e2`	multiplication	**L**	numeric	numeric
`e1 / e2`	division	**L**	numeric	numeric
`e1 % e2`	remainder	**L**	numeric	numeric
`e1 + e2`	addition	**L**	numeric	numeric
`e1 + e2`	string concatenation	**L**	String, any	String
`e1 + e2`	string concatenation	**L**	any, String	String
`e1 - e2`	subtraction	**L**	numeric	numeric
`e1 < e2`	less than	**N**	numeric	boolean
`e1 <= e2`	less than or equal to	**N**	numeric	boolean
`e1 >= e2`	greater than or equal to	**N**	numeric	boolean
`e1 > e2`	greater than	**N**	numeric	boolean
`e1 == e2`	equal	**L**	compatible	boolean
`e1 != e2`	not equal	**L**	compatible	boolean
`e1 && e2`	logical and	**L**	boolean	boolean
`e1 || e2`	logical or	**L**	boolean	boolean
`x = e`	assignment	**R**	`e` subtype of `x`	type of `x`

Expressions

Table of Expression Forms

- In the table above, operators in the same group of rows have higher precedence than those below
- The column **A** represents **operator associativity**
  - **L** left associative
    `e1 + e2 + e3` means `(e1 + e2) + e3`
  - **R** right associative
    `e1 = e2 = e3` means `e1 = (e2 = e3)`
  - **N** not associative
    `1 < x < 4` is not valid
- *integer* means `char`, `byte`, `short`, `int`, `long`, or the boxed forms Character, Byte, Short, Integer, Long
- *numeric* means integer or `float`, `double` or the boxed forms Float, Double
- The article version of these notes has a more complete table of expression forms

Expressions

Arithmetic (1)

```
/* (a) */ int x1 = 4 + 6 * 3
/* (b) */ int x2 = 6 - 3 - 2

int max = 2147483647
int min = -2147483648

/* (c) */ int x3 = max + 1
/* (d) */ int x4 = min - 1
/* (e) */ int x5 = -min
```

- Arithmetic on *small* integers follows the usual rules of operator precedence and associativity
- But beware overflow
- The integer types use signed two's complement representation (except for char)

Expressions

Arithmetic (2)

```
jshell> /* (a) */ int x1 = 4 + 6 * 3
x1 ==> 22

jshell> /* (b) */ int x2 = 6 - 3 - 2
x2 ==> 1

int max = 2147483647
int min = -2147483648

jshell> /* (c) */ int x3 = max + 1
x3 ==> -2147483648

jshell> /* (d) */ int x4 = min - 1
x4 ==> 2147483647

jshell> /* (e) */ int x5 = -min
x5 ==> -2147483648
```

Expressions

Arithmetic (3)

```
/* (f) */ int x6 = 10/3
/* (g) */ int x7 = (-10)/3
/* (h) */ int x8 = 10/(-3)
/* (i) */ int x9 = (-10)/(-3)

/* (i) */ int x10 = 10%3
/* (j) */ int x11 = (-10)%3
/* (k) */ int x12 = 10%(-3)
/* (l) */ int x13 = (-10)%(-3)
```

Expressions

Arithmetic (4)

```
jshell> /* (f) */ int x6 = 10/3
x6 ==> 3

jshell> /* (g) */ int x7 = (-10)/3
x7 ==> -3

jshell> /* (h) */ int x8 = 10/(-3)
x8 ==> -3

jshell> /* (i) */ int x9 = (-10)/(-3)
x9 ==> 3
```

Expressions

Arithmetic (5)

```
jshell> /* (i) */ int x10 = 10%3
x10 ==> 1

jshell> /* (j) */ int x11 = (-10)%3
x11 ==> -1

jshell> /* (k) */ int x12 = 10%(-3)
x12 ==> 1

jshell> /* (l) */ int x13 = (-10)%(-3)
x13 ==> -1
```

Expressions

Arithmetic (6)

- **Definitions of Division and Remainder** different languages have several different definitions
- **Java** `int x/y` is integer division truncated toward zero
- Remainder `x % y = x - (x/y) * y`
- See Boute (1992) *The Euclidean Definition of the Functions div and mod*, Modulo operation, Leijin (2003),
- The **Haskell** equivalents are `quot` and `rem`
- You may have seen a definition of `div` as integer division with truncation towards negative infinity and `mod` satisfying (Haskell notation)

```
(x `div` y) * y + (x `mod` y) == x
```

- Note that neither of the above definitions follow Euclidean division since the remainder can be negative

Expressions

Arithmetic (7)

- Below is a definition of a method, `boolean isOdd(int n)`, which is intend to return `true` if its argument is odd
- What is wrong with the function ?

```java
boolean isOdd(int n) {
  return n % 2 == 1;
}
```

Expressions

Arithmetic (8)

- The function works for positive integers but not for negative

```
jshell> boolean isOdd(int n) {
   ...>   return n % 2 == 1;
   ...> }
|  created method isOdd(int)

jshell> boolean b9P = isOdd(9)
b9P ==> true

jshell> boolean b9N = isOdd(-9)
b9N ==> false
```

Expressions

Arithmetic (9)

- Change the function to check it is not even
- Same answer whether dividend is positive or negative

```
boolean isOdd(int n) {
  return n % 2 != 0 ;
}
```

```
jshell> boolean isOdd(int n) {
   ...>   return n % 2 != 0 ;
   ...> }
|  modified method isOdd(int)

jshell> boolean b9P = isOdd(9)
b9P ==> true

jshell> boolean b9N = isOdd(-9)
b9N ==> true
```

Expressions

Arithmetic (10)

- **Floating Point Arithmetic** is not Real
- See Goldberg (1991), Floating-point arithmetic
- Which of the following are valid and if so what is the result ?

```
/* (a) */ double d1 = 10.0/3.0 ;
/* (b) */ double d2 = 10.0 % 3.0 ;
          double dm1 = -1.0 ;
/* (c) */ double d3 = 1.0/0.0 ;
/* (d) */ double d4 = 0.0/0.0 ;
/* (e) */ double d5 = 0 * d3 ;
/* (f) */ double d6 = 1 + d3 ;
/* (g) */ double d7 = d3 + d3 ;
/* (h) */ double d8 = d3 - d3 ;
/* (i) */ double d9 = d3 / d3 ;
/* (j) */ double d10 = 3 % d3 ;
/* (k) */ double d10 = 3 % 0 ;
/* (l) */ double d11 = d3 % 3 ;
/* (m) */ double d12 = Math.sqrt(dm1) ;
```

Expressions
Arithmetic (11)

- **Floating Point Arithmetic** is not Real

```
jshell> /* (a) */ double d1 = 10.0/3.0 ;
d1 ==> 3.3333333333333335

jshell> /* (b) */ double d2 = 10.0 % 3.0 ;
d2 ==> 1.0

          double dm1 = -1.0 ;
```

Expressions
Arithmetic (12)

- **Floating Point Arithmetic** is not Real

```
jshell> /* (c) */ double d3 = 1.0/0.0 ;
d3 ==> Infinity

jshell> /* (d) */ double d4 = 0.0/0.0 ;
d4 ==> NaN

jshell> /* (e) */ double d5 = 0 * d3 ;
d5 ==> NaN

jshell> /* (f) */ double d6 = 1 + d3 ;
d6 ==> Infinity
```

Expressions

Arithmetic (13)

- **Floating Point Arithmetic** is not Real

```
jshell> /* (g) */ double d7 = d3 + d3 ;
d7 ==> Infinity

jshell> /* (h) */ double d8 = d3 - d3 ;
d8 ==> NaN

jshell> /* (i) */ double d9 = d3 / d3 ;
d9 ==> NaN

jshell> /* (j) */ double d10 = 3 % d3 ;
d10 ==> 3.0
```

Expressions

Arithmetic (14)

- **Floating Point Arithmetic** is not Real

```
jshell> /* (k) */ double d10 = 3 % 0 ;
|  Exception java.lang.ArithmeticException: / by zero
|        at (#68:1)

jshell> /* (k) */ double d10 = 3.0 % 0.0 ;
d10 ==> NaN

jshell> /* (l) */ double d11 = d3 % 3 ;
d11 ==> NaN

jshell> /* (m) */ double d12 = Math.sqrt(dm1) ;
d12 ==> NaN
```

Arithmetic

Mistakes with Built-in Classes

- The first line below defines double d3 as before to be Infinity
- The next two line have mistakes — what are they ?

```
double d3 = 1.0/0.0 ;
/* (a) */ boolean bd3 = Math.isInfinite(d3) ;
/* (b) */ boolean bd3 = d3.isInfinite()
```

Arithmetic

Mistakes with Built-in Classes (2)

```
jshell> double d3 = 1.0/0.0
d3 ==> Infinity

jshell> /* (a) */ boolean bd3 = Math.isInfinite(d3)
|  Error:
|  cannot find symbol
|    symbol:   method isInfinite(double)
|  boolean bd3 = Math.isInfinite(d3);
|                ^-------------^

jshell> /* (b) */ boolean bd3 = d3.isInfinite()
|  Error:
|  double cannot be dereferenced
|  boolean bd3 = d3.isInfinite();
|                ^-----------^
```

- In (a) `isInfinite` is a method of the `Double` class
- In (b) `isInfinite` is being used as an instance method but `d3` is a primitive variable not an object reference

Arithmetic

Mistakes with Built-in Classes (3)

- **Solution** explicitly box d3 to Double is one solution

```
jshell> Double d3D = new Double(d3)
d3D ==> Infinity

jshell> boolean bd3D = d3D.isInfinite()
bd3D ==> true
```

- But the Double documentation says *It is rarely appropriate to use this constructor.*
- Automatic boxing and unboxing happens in many cases

```
jshell> Double d3D = d3
d3D ==> Infinity
```

Expressions

Logical Operators (1)

- `(==)` and `!=` compare compatible tyes — one operand must be a *subtype* of the other
- Every type is a subtype of itself — `t1` is a subtype of `t2` if any value `v1` of `t1` can be used where a value of type `t2` is expected
- Are the following valid and if so, what do they evaluate to ?

```java
/* (a) */ boolean b4 = 10 == 9
/* (b) */ boolean b5 = 10 == 10.0
/* (c) */ boolean b6 = 1.0/0.0 == 1.0/0.0
/* (d) */ boolean b7 = 0.0/0.0 == 0.0/0.0
/* (e) */ boolean b8 = 0.0/0.0 < 0.0/0.0
/* (f) */ boolean b9 = 0.0/0.0 > 0.0/0.0
```

Expressions

Logical Operators (2)

```
jshell> /* (a) */ boolean b4 = 10 == 9
b4 ==> false

jshell> /* (b) */ boolean b5 = 10 == 10.0
b5 ==> true

jshell> /* (c) */ boolean b6 = 1.0/0.0 == 1.0/0.0
b6 ==> true

jshell> /* (d) */ boolean b7 = 0.0/0.0 == 0.0/0.0
b7 ==> false

jshell> /* (e) */ boolean b8 = 0.0/0.0 < 0.0/0.0
b8 ==> false

jshell> /* (f) */ boolean b9 = 0.0/0.0 > 0.0/0.0
b9 ==> false
```

Expressions

Logical Operators (3)

- The logical operators (&&) (AND) and (||) (OR) are *lazy* in their second argument

```
jshell> boolean b10 = true || (1/0 == 1.0/0.0)
b10 ==> true

jshell> boolean b11 = (1/0 == 1.0/0.0)
|  Exception java.lang.ArithmeticException: / by zero
|        at (#83:1)
```

Classes

Overview and Structure

- A **class** represents a concept, a template for creating instances (objects)
- An **object** is an instance of a concept (a class)
- A classDeclaration of `class C` has the form

```
classModifiers class C extendsClause implementsClause
  classBody
```

- extendsClause and implementsClause refer to superclasses and interface (see later in M250)
- For a top-level class classModifiers may be a list of `public` and at most one of `abstract` or `final`

Classes

Overview and Structure (2)

- The classBody contains declarations of fields, constructors, methods, nested classes, nested interfaces, and initialiser blocks (M250 mainly uses the first three)
- The declarations *may* appear in any order but you should use the order suggested in *M250 Code Conventions*

```
{
  fieldDeclarations
    /* class (static) variables */
    /* instance variables */
  constructorDeclarations
  methodDeclarations
}
```

- A source file may begin with `package` (not used in M250) and `import` declarations (to be covered later)

Class Example

TMA01 Practice Quiz (1)

- Open BlueJ and create a new Project
- Project › New Project. . .
- There may be a problem navigating folders — in that case use the text box
- Create new class Edit › New Class. . . M250Colour

```java
/**
 * Write a description of class M250Colour here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class M250Colour
{
```

```java
}
```

Class Example

TMA01 Practice Quiz (2)

- (a)(i) Write a `private` instance field `String hexColour`
- (a)(ii) Write a constructor for `M250Colour` initialising `hexColour` to `"#000000"`

```java
    // instance variables
    private String hexColour ;

    /**
     * Constructor for objects of class M250Colour
     */
    public M250Colour()
    {
        // initialise instance variables
        hexColour = "#000000" ;
    }
```

Class Example

TMA01 Practice Quiz (3)

- (a)(iii) Write a getter method for `hexColour`

```java
  /**
   * Returns the value of hexColour of the receiver
   *
   * @return hexColour of the receiver
   */
  public String getHexColour(){
    return this.hexColour ;
  }
```

- Notice I prefer K&R layout

Class Example

TMA01 Practice Quiz (4)

- (b)(i) Write a public method `isValidLength(String hStr)` to check `hStr` has 7 characters

```java
  /**
   * Returns true if the input String has length 7
   *
   * @return true if the input String has length 7
   */
  public boolean isValidLength(String hStr){
    final int hexStrLen = 7 ;
    return hStr.length() == hexStrLen ;
  }
```

- Note alternative

```java
  public boolean isValidLength(String hStr){
    return hStr.length() == 7;
  }
```

Class Example

TMA01 Practice Quiz (5)

- (b)(ii) Write `isValidFirst(String hStr)` to check the first character is `'#'`

```java
  public boolean isValidFirst(String hStr){
    final char hexStrPrefix = '#' ;
    return hStr.length() > 0
           && (hStr.charAt(0) == hexStrPrefix) ;
  }
```

- Alternative

```java
  public boolean isValidFirst(String hStr){
    return hStr.length() > 0
           && (hStr.charAt(0) == '#');
  }
```

Class Example

TMA01 Practice Quiz (6)

- (b)(iii) Write a method `isValidCharacters(String hStr)` to check the rest of the characters are valid hex

```java
  public boolean isValidCharacters(String hStr){
    boolean validChr ;
    int hStrLen = hStr.length() ;
    char hStrCharAtI ;

    for (int i = 1 ; i <= hStrLen - 1 ; i++) {
      hStrCharAtI = hStr.charAt(i) ;
      validChr
        =  (('0' <= hStrCharAtI
             && hStrCharAtI <= '9')
            || ('A' <= hStrCharAtI
               && hStrCharAtI <= 'F')) ;
      if (!validChr) {
        return false ;
      }
    }
    return true ;
  }
```

Class Example

TMA01 Practice Quiz — Error 1(a)

- (b)(iii) What are the errors in:

```
public boolean isValidCharacters(String h){

  for (int position = 1; position < 7; position ++){
    if ((h.charAt(position) >= 0
         && h.charAt(position) <= 9)
        || (h.charAt(position) >= 'A'
            && h.charAt(position) <= 'F')){
      return true;
    }
  }
  return false;
}
```

Class Example

TMA01 Practice Quiz — Error 1(b)

1. In the code for `isValidCharacters()` there is a `for` loop with an `if` condition — if the condition is `true` for at least one character in the 6 (six) characters then the whole lot are regarded as valid — the loop will only return `false` if the `if` condition always evaluates to `false`
2. The condition is comparing a character to the values denoted by `0` and `9` and not the characters `'0'` and `'9'` — why does this not generate an error ? Because in Java characters are regarded as numeric types — so in the comparison, the character is coerced to its value as a Unicode code point — for example, `'2'` has Unicode code point 50 so is coerced to 50

Class Example

TMA01 Practice Quiz (7)

- (b)(iii) Alternative

```
public boolean isValidCharactersA(String hStr)
{
   String validValues = "0123456789ABCDEF" ;
   int hStrLen = hStr.length() ;
   String hSubStr ;

   for(int index = 1; index <= hStrLen - 1; index++)
   {
     hSubStr = hStr.substring(index, index + 1) ;
     if (!validValues.contains(hSubStr)) {
         return false ;
     }
   }
   return true ;
}
```

Class Example

TMA01 Practice Quiz (8)

- (b)(iv) Write a method isValidHexColour(String hStr) that combines the three checks

```
  public boolean isValidHexColour(String hStr){
    return isValidLength(hStr)
           && isValidFirst(hStr)
           && isValidCharacters(hStr) ;
  }
```

Class Example

TMA01 Practice Quiz (9)

- (c) Write a setter method for `M250Colour` that outputs an appropriate message

```
  public void setHexColour(String hStr){
    boolean validStr = isValidHexColour(hStr) ;
    String msg ;

    if (validStr) {
      msg = ("Colour " + hStr + " is valid") ;
      this.hexColour = hStr ;
    } else {
      msg = ("Colour " + hStr + " is not valid") ;
    }
    System.out.println(msg) ;
  }
```

Class Example

TMA01 Practice Quiz — Sample Tests 1(a)

- Not returning a boolean by incomplete expression

```
jshell>    public boolean isValidLength(String hStr){
   ...>      return hStr.length();
   ...>    }
   ...>
|  Error:
|  incompatible types: int cannot be converted to boolean
|       return hStr.length();
|              ^-----------^
```

Class Example

TMA01 Practice Quiz — Sample Tests 1(b)

- Using assignment where you meant equality test

```
jshell>    public boolean isValidLength(String hStr){
   ...>      return hStr.length() = 7;
   ...>    }
   ...>
|  Error:
|  unexpected type
|    required: variable
|    found:    value
|       return hStr.length() = 7;
|              ^------------^
```

Class Example

TMA01 Practice Quiz — Sample Tests 1(c)

- Example usage in JShell
- `mClr` is a reference to an object — it is displayed in JShell in the form `<class>@<hexDigits>`
- See Object toString() method for an explanation

```
jshell> M250Colour mClr = new M250Colour()
mClr ==> M250Colour@68de145

jshell> String str1 = mClr.getHexColour()
str1 ==> "#000000"

jshell> mClr.setHexColour("#FF0000")
Colour #FF0000 is valid

jshell> String str2 = mClr.getHexColour()
str2 ==> "#FF0000"

jshell> String str3 = mClr.getClass().getName()
          + '@' + Integer.toHexString(mClr.hashCode())
str3 ==> "M250Colour@68de145"
```

Class Example

TMA01 Practice Quiz — Sample Tests 1(d)

- Example usage in JShell
- Note that `isValidLength()`, `isValidFirst()`, `isValidCharacters()`, `isValidHexColour()` are *instance methods*

```
jshell> M250Colour mClr = new M250Colour()
mClr ==> M250Colour@306a30c7

jshell> boolean b1 = isValidLength("asdf")
|  Error:
|  cannot find symbol
|    symbol:   method isValidLength(java.lang.String)
|  boolean b1 = isValidLength("asdf");
|               ^-----------^

jshell> boolean b1 = mClr.isValidLength("asdf")
b1 ==> false
```

Class Example

TMA01 Practice Quiz — Error 2(a)

- what *might* (almost certainly) wrong with the following:

```
public boolean isValidHexColour(String h){

  if (isValidCharacters(h) == true
      && isValidFirst(h) == true
      && isValidLength(h) == true){
    return true;
  } else {
    return false;
  }
}
```

Class Example

TMA01 Practice Quiz — Error 2(b)

1. What happens if the string is empty ?
2. If the first character is not valid, it is not worth checking the rest

Class Example

TMA01 Practice Quiz — Error 3(a)

- The following does not compile — what is the error message and why ?

```
public boolean isValidCharactersA(String h) {
  for (int i = 1; i < 8; i++) {
    if (!(   (h.charAt(i)>= 48
              && h.charAt(i) <= 57)
          || (h.charAt(i) >= 65
              && h.charAt(i) <= 70))) {
      return false;
    }
    return true;
  }
}
```

Class Example

TMA01 Practice Quiz — Error 3(b)

- Here is the error message — but why ?

```
jshell> public boolean isValidCharactersA(String h) {
   ...>   for (int i = 1; i < 8; i++) {
   ...>     if (!(   (h.charAt(i)>= 48
   ...>               && h.charAt(i) <= 57)
   ...>            || (h.charAt(i) >= 65
   ...>               && h.charAt(i) <= 70))) {
   ...>       return false;
   ...>     }
   ...>     return true;
   ...>   }
   ...> }
   ...>
|  Error:
|  missing return statement
|  public boolean isValidCharactersA(String h) {
|                                              ^
```

Class Example

TMA01 Practice Quiz — Error 3(c)

- If a method is declared to have a return type, then the method must return a value — it must not be possible for execution to reach the end of a method body without executing a `return` statement (see *Java Language Specification (JLS)* Section 8.4.7 (Edition 13) *Method Body* for full details, but a bit formal)
- Why is the compiler saying *Missing return* when we can see two and the code is bound to hit one ?
- The compiler has to work for *every* syntactically valid program so it has to have some *effectively computable* rules
- We go back to *Java Language Specification (JLS)* Section 14.21 (Edition 13) *Unreachable Statements* and try and work out what the Java compiler is expected to do with *for* statements

Class Example

TMA01 Practice Quiz — Error 3(d)

- Essentially a *for* statement *can complete normally* if the statement is reachable and the condition is not a constant `true`
- So in terms of program flow the compiler doesn't know whether the loop terminates or not
- The analysis of the compiler is a syntactic check on where the program execution could go
- to work out whether an arbitrary block of code or statement would or would not terminate is equivalent to solving the Halting problem which we know is not solvable (see M269)
- So the code is missing a `return` statement *after* the `for` loop
- However, if the compiler had accepted the code, then it would still have returned `true` if the first character was valid

Java Shell, JShell

References

- JShell is a Java *read-eval-print loop (REPL)* introduced in 2017 with JDK 9
- Java Shell User's Guide (Release 12, March 2019)
- Tools Reference: jshell
- JShell Tutorial (30 June 2019)
- How to run a whole Java file added as a snippet in JShell? (15 July 2019)

What Next ?

Programming, Debugging, Psychology

Although programming techniques have improved immensely since the early days, the process of finding and correcting errors in programming — known graphically if inelegantly as *debugging* — still remains a most difficult, confused and unsatisfactory operation. The chief impact of this state of affairs is psychological. Although we are happy to pay lip-service to the adage that to err is human, most of us like to make a small private reservation about our own performance on special occasions when we really try. It is somewhat deflating to be shown publicly and incontrovertibly by a machine that even when we do try, we in fact make just as many mistakes as other people. If your pride cannot recover from this blow, you will never make a programmer.

Christopher Strachey, Scientific American 1966 vol 215 (3) September pp112–124

What Next ?

To err is human ?

- To err is human, to really foul things up requires a computer.
- Attributed to Paul R. Ehrlich in 101 Great Programming Quotes
- Attributed to Bill Vaughn in Quote Investigator
- Derived from Alexander Pope (1711, An Essay on Criticism)
- *To Err is Humane; to Forgive, Divine*
- This also contains

  > *A little learning is a dangerous thing;*
  > *Drink deep, or taste not the Pierian Spring*

- In programming, this means you have to *read the fabulous manual* (RTFM)

What Next ?

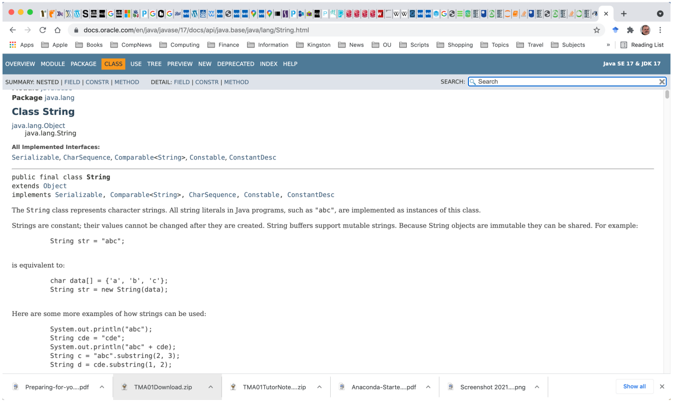
Chps 1–4, TMA01

- Chps 4–5, Iteration, collections; Functional Java (optional)
- Tutorial 10:00 Sunday 16 November 2025 online
- TMA01 Thursday 11 December 2025

- Java Documentation — BlueJ has JDK 7 embedded, JDK 13 is current (2019)
- JDK 13 Documentation
- Java Platform API Specification
- Java Language Specification
- JDK Documentation 〉API Documentation 〉java.base
  - java.lang — fundamental classes for the Java programming language
  - java.util — Collections framework

Java

API Documentation (1)



docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html
Apps Apple Books CompNews Computing Finance Information Kingston News OU Scripts Shopping Topics Travel Subjects » Reading List
OVERVIEW MODULE PACKAGE CLASS USE TREE PREVIEW NEW DEPRECATED INDEX HELP
Java SE 17 & JDK 17
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD
SEARCH: Search
Preparing-for-yo....pdf
TMA01Download.zip
TMA01TutorNote....zip
Anaconda-Starte....pdf
Screenshot 2021....png
Show all

Package java.lang

Class String

java.lang.Object
    java.lang.String

All Implemented Interfaces:

`Serializable, CharSequence, Comparable<String>, Constable, ConstantDesc`

```
public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence, Constable, ConstantDesc
```

The `String` class represents character strings. All string literals in Java programs, such as `"abc"`, are implemented as instances of this class.

Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:

```
     String str = "abc";
```

is equivalent to:

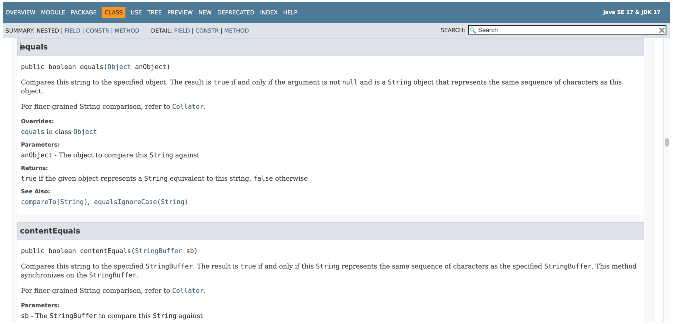
```
     char data[] = {'a', 'b', 'c'};
     String str = new String(data);
```

Here are some more examples of how strings can be used:

```
     System.out.println("abc");
     String cde = "cde";
     System.out.println("abc" + cde);
     String c = "abc".substring(2, 3);
     String d = cde.substring(1, 2);
```

- `Strings` are *immutable* objects
- See java.lang.StringBuilder for *mutable* strings
- In a *functional programming approach* everything is immutable — it makes life simpler (but at a cost)

Java
API Documentation (2)


OVERVIEW MODULE PACKAGE CLASS USE TREE PREVIEW NEW DEPRECATED INDEX HELP
Java SE 17 & JDK 17
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD
SEARCH: Search

equals

```
public boolean equals(Object anObject)
```

Compares this string to the specified object. The result is `true` if and only if the argument is not `null` and is a `String` object that represents the same sequence of characters as this object.

For finer-grained String comparison, refer to `Collator`.

Overrides:
`equals` in class `Object`

Parameters:
`anObject` - The object to compare this `String` against

Returns:
`true` if the given object represents a `String` equivalent to this string, `false` otherwise

See Also:
`compareTo(String)`, `equalsIgnoreCase(String)`

contentEquals

```
public boolean contentEquals(StringBuffer sb)
```

Compares this string to the specified `StringBuffer`. The result is `true` if and only if this `String` represents the same sequence of characters as the specified `StringBuffer`. This method synchronizes on the `StringBuffer`.

For finer-grained String comparison, refer to `Collator`.

Parameters:
`sb` - The `StringBuffer` to compare this `String` against

- Remember (==) tests for *identity* — what does this mean ?

M250

Books Phil Likes

- M250 is self contained — you do not need further books — but you might like to know about some:
- Sestoft (2016) *Java Precisely* — the best short reference
- Evans, Flanagan (2018) *Java in a Nutshell* — the best longer reference
- Barnes, Kölling (2016) *Objects First with Java* — the BlueJ book — see www.bluej.org for documentation and tutorial
- Bloch (2017) *Effective Java* — guide to best practice