Working Memo :)

11910738 刘颖潼

```
public void start( Stage, stage ) {
    Label     文本标签
    Button    按钮 → setonAction (监听点击 (松开起效))
    Platform.exit() → make the application terminate.
```

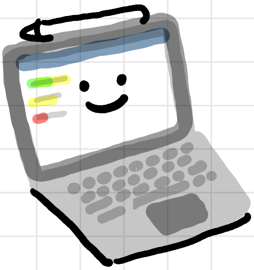
HelloWorldFX.java

⇓

设置 HBox 有指定间距的布局. (在HelloWorld里装按钮)
BorderPane 也是大布局

⇓

构造 Scene (BP...Width, Height); stage.show() ⇒ 画面出现!
stage.setScene (..)
}



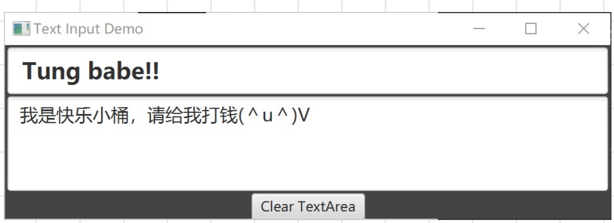
用到了 Canvas (画布)

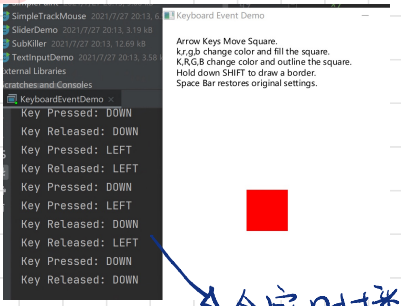
HighLowGUI.java

实现了一个小游戏 (Higher \ Lower)
主对 Hand (手牌), Message. 操作
Deck
start 部分差不多, 设置按键位置与启动.
调用 drawBoard 更新已处理卡牌. (每次click)
drawCard
结束后 (gameInProgress == False) 可重开游戏

TextInputDemo.java

文本输入器
javafx.TextField (作标题栏)
TextArea (作正文栏)
setPrefColumn\RowCount 可设长宽.





KeyBoardEvent.java

scene.setOnKeyPressed 按
Released 松
Typed (一些特别事件)

KeyCode key = evt.getCode (获取).

会实时播报按键

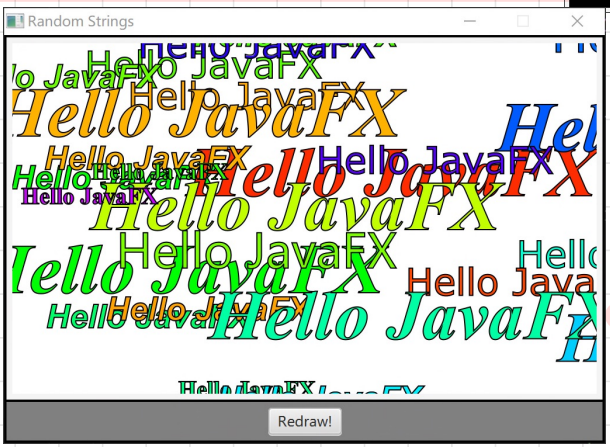
MosaicDraw.java

可以通过鼠标画马赛克图案.
拖拽 Dragged
按下 Pressed

菜单 Menu Bar

Radio 单选题

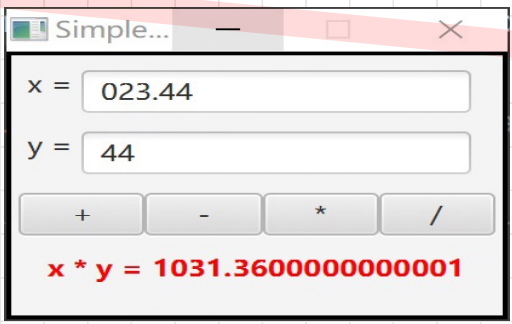
isSelected (Boolean)
→ 作相应操作.



RandomString.java

和 RandomCard 一样用 Math.random 去获取字体格式 :

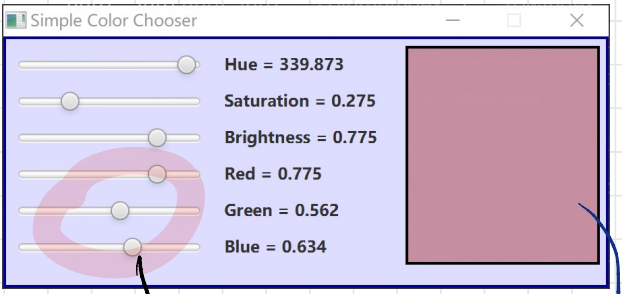
Font.font(...)
大小, 色, 字体...



SimpleCalc.java 计算器

TextField Label

按键控制



ColorChooser.java

我喜欢这个!

Color属性(R, G, B, Hue, Bright, Saturation).

每次改变产生新色

Slider 滑动条

(min, max, value)

Pane

需手动加监听 AddListener(...)

SimplePaint.

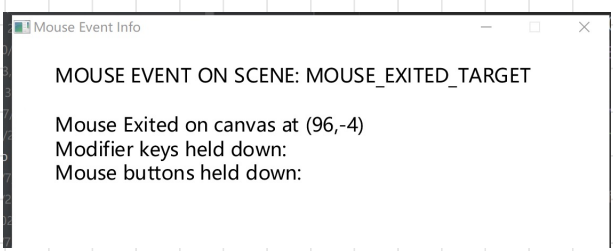
Canvas 画布

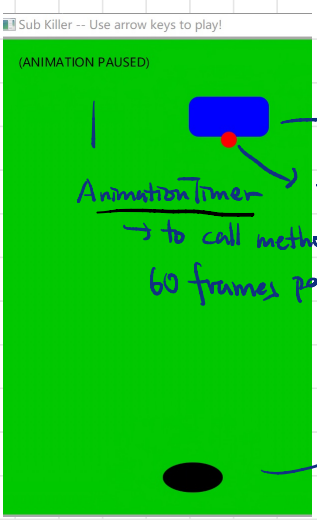
GraphicsContext (画画用)

画: 随鼠标

TrackMouse.java ⇨

跟踪鼠标活动并打印





SubKiller.java.

Boat

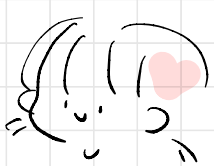
Bomb

AnimationTimer

→ to call method.

60 frames per second.

复杂一点的小游戏



Canvas

Stage.

keycode (获取键盘)

移动随机(random)

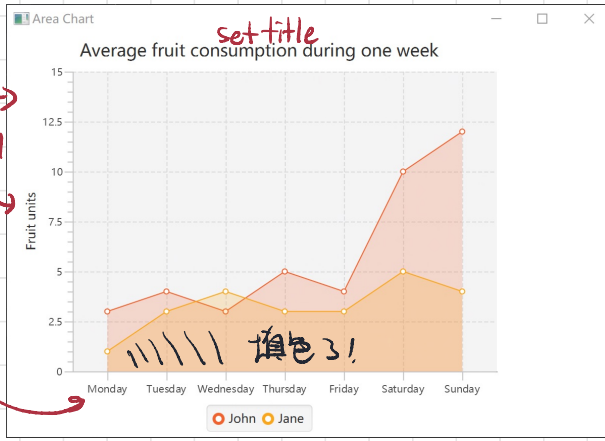
向左右

Submarine

Working Memo

11910738 刘欣潼 Java2

AreaChartExample.java



set title

set label

NumberAxis 数值轴

CategoryAxis 实现一条轴 (字符串的度量)

AreaChart 大图表 (X轴, Y轴)

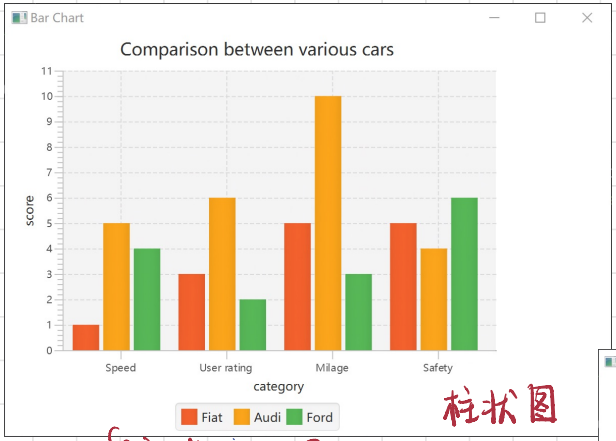
XYChart.Series 可用于设数据

.getData().add 加数据.

再创 Group, Scene, Stage 装下它们.

填色折线图

BarChartExample.java



柱状图

Series.Name.

同样使用 数轴, 字符轴.

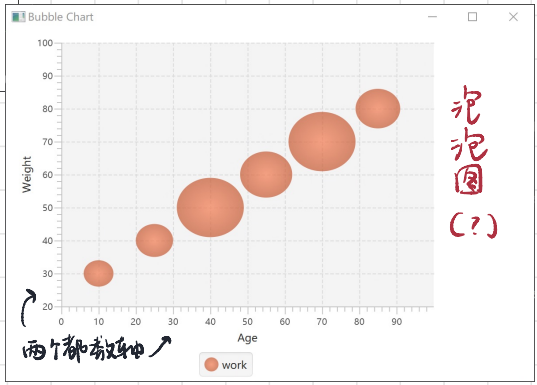
BarChart<XA, YA> 柱状图类

同样对 series.getData.add() 加数据.

BubbleChart.java

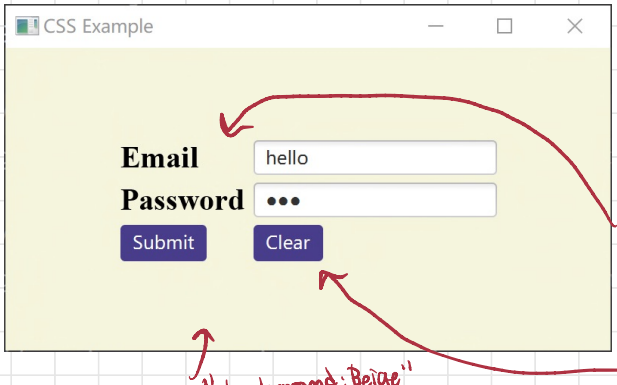
BubbleChart 为泡泡图类.

不同的是, series除了x,y值, 还有extraValue (用处?)



泡泡图 (?)

两个都数轴



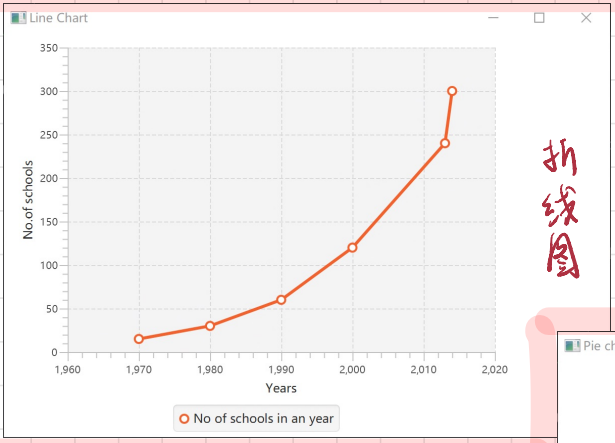
CssExample.java.

☆. CSS is not available on IDEA community but ultimate

setStyle("-fx-font: normal bold 20px 'serif'")

("-fx-background-color:darkslateblue;
-fx-text-fill:white;")

可以轻易修改颜色(不用.css文件下)

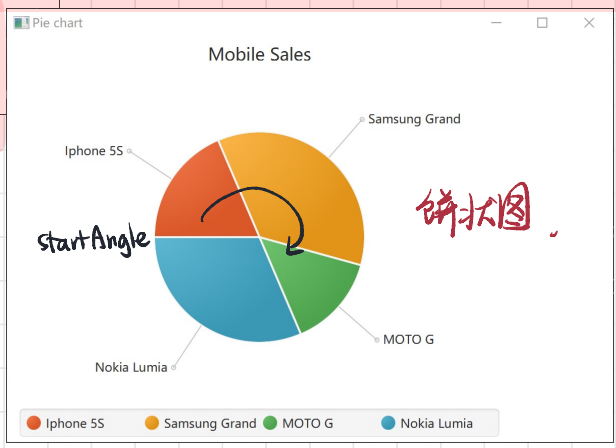


折线图

LineChartExample.java

与上面AreaChart不同
它下面不会填充Area

其它两个一致.



饼状图.

PieChartExample.java

饼状图表 PieChart

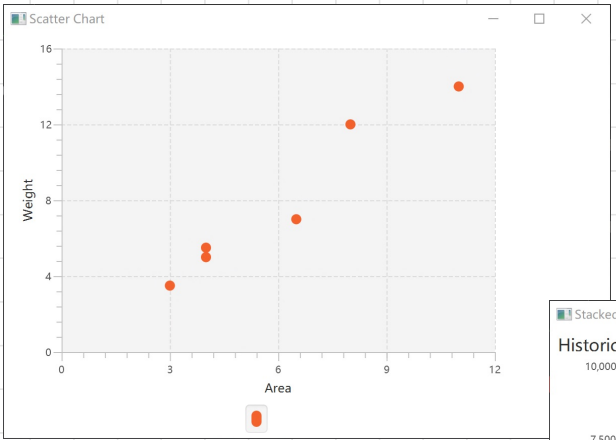
不同的是,数据不是series.

而是

```
ObservableList<PieChart.Data> pieChartData =
        FXCollections.observableArrayList(
          new PieChart.Data("Iphone 5S", 13),
          new PieChart.Data("Samsung Grand", 25),
          new PieChart.Data("MOTO G", 10),
          new PieChart.Data("Nokia Lumia", 22)
        );

//Creating a Pie chart
PieChart pieChart = new PieChart(pieChartData);
```

添加

还需设置 setClockwise,
setStartAngle.

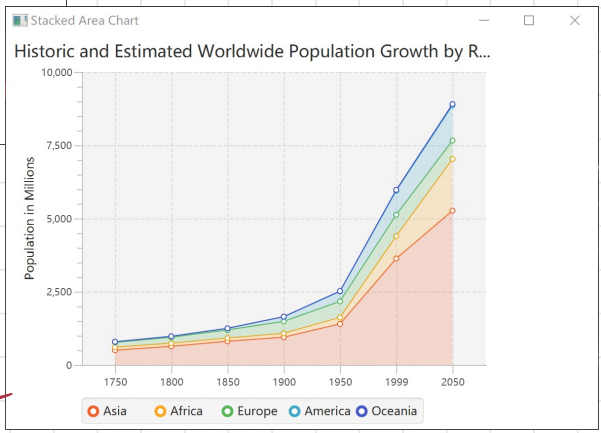


ScatterChartExample.java.

散点图.

数据为series. 再加到折线图

以 getData.add 加点.

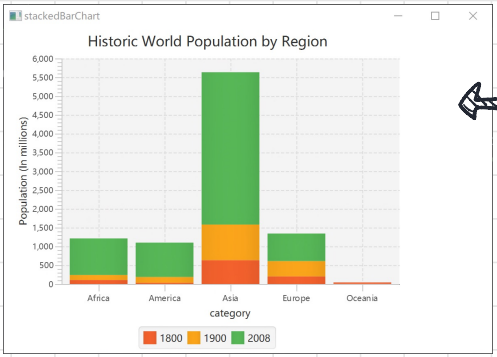


StackedAreaExample.

```
CategoryAxis xAxis = new CategoryAxis();

xAxis.setCategories(FXCollections.<String>observableArrayList(
    Arrays.asList("1750", "1800", "1850", "1900", "1950", "1999",
"2050" )));
```

x轴的值加上

与第一个没太大差别. 但有bug



堆叠柱状图 Stacked BarChart.

只需在 BarChart 大类下

以 getData.addAll(series1, series2 ...)

就可以各自在一个图中.

FitPlot.java 拟合图

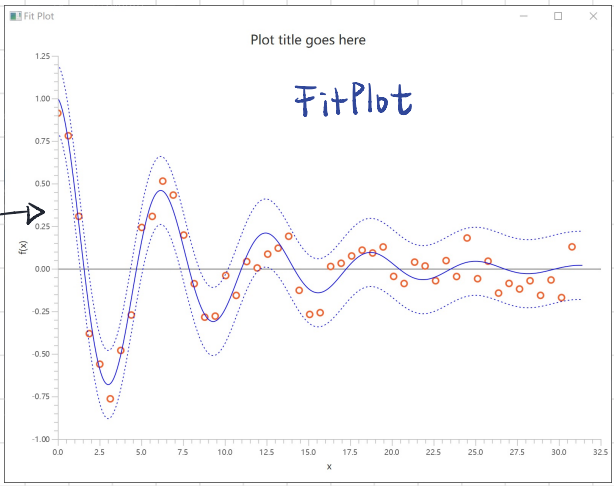
先随机设3几个数据点

设立 Sigma, Damp Sigma bounds

```
int nSim = 1000;
double xSim[] = new double[nSim];
double ySim[] = new double[nSim];
double ySimUp[] = new double[nSim];
double ySimDn[] = new double[nSim];
for (int i = 0; i < nSim; i++) {
    xSim[i] = i * 5*2*Math.PI / nSim;
    ySim[i] = Math.exp(-xSim[i]*damp*damp/2) * Math.cos(xSim[i]);
    ySimUp[i] = ySim[i] + 2*sigma;
    ySimDn[i] = ySim[i] - 2*sigma;
}
```

拟合的数

数据为 series.



Review Memo of Java2 CS209A

主要内容 ·数据处理 · GUI实现 · 流处理 ·程序评估 · regexp
·一些先进的程序技术

- [ ] Topic 1: Computing, Basics of Programming: Types, Matrix, I/O Basics
- [ ] Topic 2: OOP, Exceptions, i18n & Characters, File I/O
- [ ] Topic 3: Basics of Images & Graphics
- [ ] Topic 4: GUI and Basics of JavaFX
- [ ] Topic 5: Generic Types & Collections
- [ ] Topic 6: Lambda Functions & Stream Programming
- [ ] Topic 7: File Formats & Data Visualization
- [ ] Topic 8: Events Handling & more JavaFX
- [ ] Topic 9: Regular Expression, Text Processing & Web Crawlers
- [ ] Topic 10: Correctness, Robustness & Efficiency
- [ ] Topic 11: More on Java Types, Reflection & JVM
- [ ] Topic 12: Refactoring & Design Patterns
- [ ] Topic 13: Multi-Thread Programming & Web Servers (optional)

Lec1 计算，编程基础：内置类，数组，循环 和 Basic I/O

Computing 发生于信息系统

软件 = 电脑运行的，能处理数据文件的程序
Software　　program

一系列指定操作的指令

Memory 存数据
CPU 执行指令

也有寄存器.

冯诺依曼结构中，内存分为 可执行的指令(code)，静态数据，基本类数据的栈和创建对象的堆.

Java虚拟机：可分为三个组成：① 类加载器，② 运行时数据区 ③ 报错引擎

基本类型 primitive types：char, boolean, int, long, float, double, byte, short [new不能用于基本类型]

数组：创建数组时，Java在内存中保留足够的空间存储元素，叫内存分配.（声明后类型,维不可改）
（从0开始计 index）（基类 初始值一般为 默认）（可用于处理矩阵）args：命令行参数

A local variable in a method can not be public.

Lec2: I/O, Complexity of CSDA, OOP, Exceptions.

软件的复杂：① 规模越大，要素越多 ② 需求定义困难 ③ 不同设计、同功能

程序设计2个重点挑战① 如何将系统分解为模块 (Decompose)

② 如何管理开发过程中的变更

For ①：高内聚，低耦合，信息隐藏

High Cohesion Loose Coupling Information Hiding

对象具有状态与行为.

一个Java文件只能定义一个public class.

AccessModifier	whenver class is accesible	extended	package	in class
Public	✓	✓	✓	✓
protected		✓	✓	✓
default			✓	✓
private				✓

Inheritance 继承 A extends B (属于关系)

□ 可视继承 □ 接口继承 □ 实现继承

It is the process where one object acquires properties of another

Polymorphism is the ability of an object to take on many forms

Polymorphism 多态

□ 覆盖 子类重新定义虚函数

□ 重载 参数不同的同名方法 ↳有多种

往下编写

抽象类中.

抽象方法只声明，无定义 {} ✗

非抽象方法有声明有定义 {}

Abstract Class

MyClass extends Ab-C..

抽象类

有抽方一定是抽类

抽象类不可实例化！

↓

AC a = new AC(); ✗

AC b = new MC(); ✓ 多态

MC c = new MC(); ✓

多态下 定义超类的抽象方法.

重写的可见性不可收窄

可定义自己的方法或实例变量

接口

extends

```java
public interface TheInterface {
   type1 methodA (Typex v1...);  // 缺省 public abstract
   type2 methodB (Typey p1...);  //
   ...
}

// Java 的 interface 即是纯抽象类，可以implements多个接口，(只能实现一个类)

public class MyClass implements TheInterface {
   public type1 methodA (Typex v1...) { ... }  // 定义接口中声明的抽象方法，+ public
   public type2 methodB (Typey p1...) { ... }  // 定义(重写)的方法可见性不能收窄
   ...
     // 可定义自己的方法或实例变量
}

TheInterface v1 = new TheInterface();  // wrong, 接口不能实例化
TheInterface v1 = new TheInterface() { ...接口方法实现代码... };    // ok, 匿名对象
TheInterface v2 = new MyClass();   // ok, 可实现多态
MyClass v3 = new MyClass();  // ok
```

一个类不能继承多个类
但是一个接口可能继承多个接口

An unchecked exception in Java derives from RuntimeException or its children

Exceptions 报错！ ⇒ Detecting || Handling

是特殊的 Objects. throw前需要new它

throw new IllegalArgumentException("xxx");

种类：1) Compile Time { ① wrong type ② Syntax Error
2) Link Time 加载
3) Run Time { ① Logic

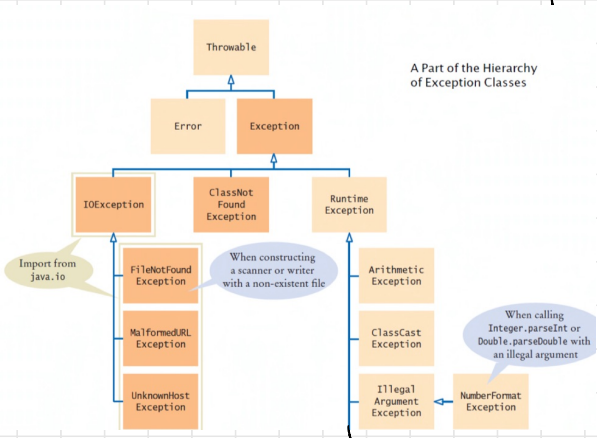
Detected by { Application / Library / OS \ Hardware

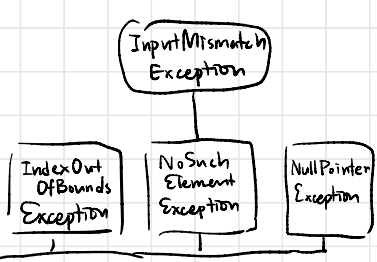
* 如果有的 error 出现得频繁，可以在方法上 throws xx Exception 来声明
* try是用来detect Exception的
* finally 会执行！

(而非抛出)

Exception：提供处理，恢复的方法

Error：严重，不可恢复。





Data Type: a set of values and a set of operations on those values

ADT: Abstract Data Type : A DT whose represen

Lec3

Interned (多路共用OS)

A string object ⟹ String s1 = "A" (Interned)

s2 = new String("A")

s3 = "A"

s1 == s3 ✓

s1 != s2

Lec3有一些变形的

Lec4

△Bitwise Operators

>>> 无脑号右移 ~取反 ^异或 &按位与 >> 右移 << 左移 0x... 字面值 十六进制

Operator	Name	Associativity
()	Parentheses	Left to right
()	Function call	Left to right
[]	Array subscript	Left to right
.	Object member access	Left to right
++	Postincrement	Left to right
--	Postdecrement	Left to right
++	Preincrement	Right to left
--	Predecrement	Right to left
+	Unary plus	Right to left
-	Unary minus	Right to left
!	Unary logical negation	Right to left
(type)	Unary casting	Right to left
new	Creating object	Right to left
*	Multiplication	Left to right
/	Division	Left to right
%	Remainder	Left to right
+	Addition	Left to right
-	Subtraction	Left to right
<<	Left shift	Left to right
>>	Right shift with sign extension	Left to right
>>>	Right shift with zero extension	Left to right
<	Less than	Left to right
<=	Less than or equal to	Left to right
>	Greater than	Left to right
>=	Greater than or equal to	Left to right
instanceof	Checking object type	Left to right
==	Equal comparison	Left to right
!=	Not equal	Left to right
&	(Unconditional AND)	Left to right
^	(Exclusive OR)	Left to right
|	(Unconditional OR)	Left to right
&&	Conditional AND	Left to right
||	Conditional OR	Left to right
?:	Ternary condition	Right to left
=	Assignment	Right to left
+=	Addition assignment	Right to left
-=	Subtraction assignment	Right to left
*=	Multiplication assignment	Right to left
/=	Division assignment	Right to left
%=	Remainder assignment	Right to left
op=	Assignment with binary operator (*op* is one of +, -, *, /, %, &, |, ^, <<, >>, >>>)	Right to left

Interger.toString(number..., 进制)

(100, 2) → 1100100

Collection是 a class under java.util which is parent interface of various collection structures.

Collections —— which contains various static methods for collection operations.

Collection 接口

List Set SortedSet

数据结构

Both Arraylist and LL implement the List Interface.

链表

List is an ordered interface, so we can control precisely where each element is inserted when using the interface

Hash map implement the Map interface.

图 Map

Map provides a mapping from key to value and Map cannot contain the same key several times, each key can only map a value

Table 1 The Methods of the Collection Interface

`Collection<String> coll = new ArrayList<>();`	The `ArrayList` class implements the `Collection` interface.
`coll = new TreeSet<>();`	The `TreeSet` class (Section 15.3) also implements the `Collection` interface.
`int n = coll.size();`	Gets the size of the collection. `n` is now `0`.
`coll.add("Harry");` `coll.add("Sally");`	Adds elements to the collection.
`String s = coll.toString();`	Returns a string with all elements in the collection. `s` is now `[Harry, Sally]`.
`System.out.println(coll);`	Invokes the `toString` method and prints `[Harry, Sally]`.
`coll.remove("Harry");` `boolean b = coll.remove("Tom");`	Removes an element from the collection, returning `false` if the element is not present. `b` is false.
`b = coll.contains("Sally");`	Checks whether this collection contains a given element. `b` is now `true`.

`for (String s : coll)`
`{`
`  System.out.println(s);`
`}`
	You can use the "for each" loop with any collection. This loop prints the elements on separate lines.

`Iterator<String> iter = coll.iterator();`	You use an iterator for visiting the elements in the collection (see Section 15.2.3).

Table 2 Working with Linked Lists

`LinkedList<String> list = new LinkedList<>();`	An empty list.
`list.addLast("Harry");`	Adds an element to the end of the list. Same as `add`.
`list.addFirst("Sally");`	Adds an element to the beginning of the list. `list` is now `[Sally, Harry]`.
`list.getFirst();`	Gets the element stored at the beginning of the list; here `"Sally"`.
`list.getLast();`	Gets the element stored at the end of the list; here `"Harry"`.
`String removed = list.removeFirst();`	Removes the first element of the list and returns it. `removed` is `"Sally"` and `list` is `[Harry]`. Use `removeLast` to remove the last element.
`ListIterator<String> iter = list.listIterator()`	Provides an iterator for visiting all list elements (see Table 3 on page 698).

Table 5 Working with Maps

`Map<String, Integer> scores;`	Keys are strings, values are `Integer` wrappers. Use the interface type for variable declarations.
`scores = new TreeMap<>();`	Use a `HashMap` if you don't need to visit the keys in sorted order.
`scores.put("Harry", 90);` `scores.put("Sally", 95);`	Adds keys and values to the map.
`scores.put("Sally", 100);`	Modifies the value of an existing key.
`int n = scores.get("Sally");` `Integer n2 = scores.get("Diana");`	Gets the value associated with a key, or `null` if the key is not present. `n` is 100, `n2` is `null`.
`System.out.println(scores);`	Prints `scores.toString()`, a string of the form `{Harry=90, Sally=100}`

`for (String key : scores.keySet()) {`
`  Integer value = scores.get(key);`
`}`
	Iterates through all map keys and values.

`scores.remove("Sally");`	Removes the key and value.

集

Set

A collection of elements which cannot contain duplicates

Table 4 Working with Sets

`Set<String> names;`	Use the interface type for variable declarations.
`names = new HashSet<>();`	Use a `TreeSet` if you need to visit the elements in sorted order.
`names.add("Romeo");`	Now `names.size()` is 1.
`names.add("Fred");`	Now `names.size()` is 2.
`names.add("Romeo");`	`names.size()` is still 2. You can't add duplicates.
`if (names.contains("Fred"))`	The `contains` method checks whether a value is contained in the set. In this case, the method returns `true`.
`System.out.println(names);`	Prints the set in the format `[Fred, Romeo]`. The elements need not be shown in the order in which they were inserted.

`for (String name : names) {`
`  . . .`
`}`
	Use this loop to visit all elements of a set.

`names.remove("Romeo");`	Now `names.size()` is 1.
`names.remove("Juliet");`	It is not an error to remove an element that is not present. The method call has no effect.

Table 8 Working with Queues

`Queue<Integer> q = new LinkedList<>();`	The `LinkedList` class implements the `Queue` interface.

`q.add(1);`
`q.add(2);`
`q.add(3);`
	Adds to the tail of the queue; `q` is now `[1, 2, 3]`.

`int head = q.remove();`	Removes the head of the queue; `head` is set to 1 and `q` is `[2, 3]`.
`head = q.peek();`	Gets the head of the queue without removing it; `head` is set to 2.

Table 7 Working with Stacks

`Stack<Integer> s = new Stack<>();`	Constructs an empty stack.

`s.push(1);`
`s.push(2);`
`s.push(3);`
	Adds to the top of the stack; `s` is now `[1, 2, 3]`. (Following the `toString` method of the `Stack` class, we show the top of the stack at the end.)

`int top = s.pop();`	Removes the top of the stack; `top` is set to 3 and `s` is now `[1, 2]`.
`head = s.peek();`	Gets the top of the stack without removing it; `head` is set to 2.

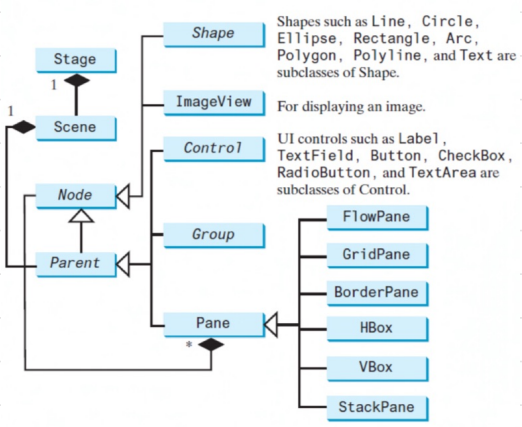
! s.empty() ↘ s.size() > 0

Lambda Expressions

xx.addListener(event -> sout("...."));

Interface name	Arguments	Returns	Example
`Predicate<T>`	`T`	`boolean`	Has this album been released yet?
`Consumer<T>`	`T`	`void`	Printing out a value
`Function<T,R>`	`T`	`R`	Get the name from an `Artist` object
`Supplier<T>`	None	`T`	A factory method
`UnaryOperator<T>`	`T`	`T`	Logical not (`!`)
`BinaryOperator<T>`	`(T, T)`	`T`	Multiplying two numbers (`*`)

Lec5 GUI and JavaFX



Nodes can be shapes, image views, UI controls, groups, and panes.

Quiz 1 10.26

1. Any java program starts running from a static driver method called __G__. main
2. All programs can be written in terms of three types of control structures: sequence, __S__ selection and repetition.
3. The four types of bugs (errors) for programs are syntax, runtime, __L__ semantics and performance.
4. In JavaFX, a __J__ Scene object is a hierarchical data structure that contains the contents of a window that is made up of "nodes" that represent visible components in the window, such as buttons and text-input boxes.
5. A(An) __E__ data type refers to a set of values and a set of operations applied on those values.
6. A(An) __A__ ADT refers to a data type whose representation is hidden from the client.
7. When a(an) __C__ checked exception may be thrown in a method body, it should be either caught in a try-catch statement in the method or defined a throws clause for the method.
8. The __D__ continue statement, when executed in a repetition statement, skips the remaining statements in the loop body and proceeds with the next iteration of the loop.

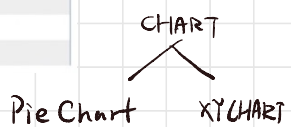
True or False

Determine whether each statement is **true(T)** or **false(F)** in Java according to its correctness.

1. Late (dynamic) binding is an essential mechanism for polymorphism.
2. In try-with-resources structure, all the resource classes should implements **AutoClosable** interface that has a method **clear()** to clear the resources. close()
3. After having the try-with-resource structure, normally a finally clause is not needed for a try statements.
4. Regular expression **[ab|mn|xy]*** matches a String with any times of combination of **ab, mn** and **xy** at least two letters.
5. A class has always at least one constructor (possibly automatically supplied by the java compiler).
6. Garbage collection in java context refers to that when all references to an object are gone, the memory used by the object is automatically reclaimed(回收).
7. Reusability is not a quality software goal in object-oriented class design.
8. Using visitor pattern, developers can add new methods for a hierarchy with fixed number of sub-type elements without change original programs.

lec6 Data Visualization with JavaFX

Class	Description
AreaChart<X,Y>	AreaChart - Plots the area between the line that connects the data points and the 0 line on the Y axis.
AreaChartBuilder<X,Y,B extends AreaChartBuilder<X,Y,B>>	Builder class for javafx.scene.chart.AreaChart
Axis<T>	Base class for all axes in JavaFX that represents an axis drawn on a chart area.
Axis.TickMark<T>	TickMark represents the label text, its associated properties for each tick along the Axis.
AxisBuilder<T,B extends AxisBuilder<T,B>>	Builder class for javafx.scene.chart.Axis
BarChart<X,Y>	A chart that plots bars indicating data values for a category.
BarChartBuilder<X,Y,B extends BarChartBuilder<X,Y,B>>	Builder class for javafx.scene.chart.BarChart
BubbleChart<X,Y>	Chart type that plots bubbles for the data points in a series.
BubbleChartBuilder<X,Y,B extends BubbleChartBuilder<X,Y,B>>	Builder class for javafx.scene.chart.BubbleChart
CategoryAxis	A axis implementation that will works on string categories where each value as a unique category(tick mark) along the axis.
CategoryAxisBuilder	Builder class for javafx.scene.chart.CategoryAxis
Chart	Base class for all charts.
ChartBuilder<B extends ChartBuilder<B>>	Builder class for javafx.scene.chart.Chart
LineChart<X,Y>	Line Chart plots a line connecting the data points in a series.
LineChartBuilder<X,Y,B extends LineChartBuilder<X,Y,B>>	Builder class for javafx.scene.chart.LineChart
NumberAxis	A axis class that plots a range of numbers with major tick marks every "tickUnit".
NumberAxis.DefaultFormatter	Default number formatter for NumberAxis, this stays in sync with auto-ranging and formats values appropriately.
NumberAxisBuilder	Builder class for javafx.scene.chart.NumberAxis
PieChart	Displays a PieChart.
PieChart.Data	PieChart Data Item, represents one slice in the PieChart
PieChartBuilder<B extends PieChartBuilder<B>>	Builder class for javafx.scene.chart.PieChart
ScatterChart<X,Y>	Chart type that plots symbols for the data points in a series.
ScatterChartBuilder<X,Y,B extends ScatterChartBuilder<X,Y,B>>	Builder class for javafx.scene.chart.ScatterChart
StackedAreaChart<X,Y>	StackedAreaChart is a variation of AreaChart that displays trends of the contribution of each value.
StackedAreaChartBuilder<X,Y,B extends StackedAreaChartBuilder<X,Y,B>>	Builder class for javafx.scene.chart.StackedAreaChart
StackedBarChart<X,Y>	StackedBarChart is a variation of BarChart that plots bars indicating data values for a category.
StackedBarChartBuilder<X,Y,B extends StackedBarChartBuilder<X,Y,B>>	Builder class for javafx.scene.chart.StackedBarChart
ValueAxis<T extends java.lang.Number>	A axis who's data is defined as Numbers.
ValueAxisBuilder<T extends java.lang.Number,B extends ValueAxisBuilder<T,B>>	Builder class for javafx.scene.chart.ValueAxis
XYChart<X,Y>	Chart base class for all 2 axis charts.
XYChart.Data<X,Y>	A single data item with data for 2 axis charts
XYChart.Series<X,Y>	A named series of data items
XYChartBuilder<X,Y,B extends XYChartBuilder<X,Y,B>>	Builder class for javafx.scene.chart.XYChart



FXCollections is a class that contains wrapper methods for regular collections so that you can use them as backend to some JavaFX widgets.

Lec7 Stream Processing

Pipe-Filter Design Patterns.

Streams 分为
I/O Streams
Byte/char
Streams

Table 1 Producing Streams

Example	Result
`Stream.of(1, 2, 3)`	A stream containing the given elements. You can also pass an array.
`Collection<String> coll = . . .;` `coll.stream()`	A stream containing the elements of a collection.
`Files.lines(path)`	A stream of the lines in the file with the given path. Use a try-with-resources statement to ensure that the underlying file is closed.
`Stream<String> stream = . . .;` `stream.parallel()`	Turns a stream into a parallel stream.
`Stream.generate(() -> 1)`	An infinite stream of ones (see Special Topic 19.1).
`Stream.iterate(0, n -> n + 1)`	An infinite stream of `Integer` values (see Special Topic 19.1).
`IntStream.range(0, 100)`	An `IntStream` of `int` values between `0` (inclusive) and `100` (exclusive)—see Section 19.8.
`Random generator = new Random();` `generator.ints(0, 100)`	An infinite stream of random `int` values drawn from a random generator—see Section 19.8.
`"Hello".codePoints()`	An `IntStream` of code points of a string—see Section 19.8.

Table 2 Collecting Results from a `Stream<T>`

Example	Comments
`stream.toArray(T[]::new)`	Yields a `T[]` array.
`stream.collect(Collectors.toList())` `stream.collect(Collectors.toSet())`	Yields a `List<T>` or `Set<T>`.
`stream.collect(Collectors.joining(", ")`	Yields a string, joining the elements by the given separator. Only for `Stream<String>`.
`stream.collect(Collectors.groupingBy(`*keyFunction*`,` *collector*`)`	Yields a map that associates group keys with collected group values—see Section 19.9.

`stream.filter(`*condition*`)`	A stream with the elements matching the condition.
`stream.map(`*function*`)`	A stream with the results of applying the function to each element.

`stream.mapToInt(`*function*`)`
`stream.mapToDouble(`*function*`)`
`stream.mapToLong(`*function*`)`
	A primitive-type stream with the results of applying a function with a return value of a primitive type—see Section 19.8.

`stream.limit(n)` `stream.skip(n)`	A stream consisting of the first `n`, or all but the first `n` elements.

`stream.distinct()`
`stream.sorted()`
`stream.sorted(`*comparator*`)`
	A stream of the distinct or sorted elements from the original stream.

Table 5 Computing Results from a Stream<T>

Example	Comments
`stream.count()`	Yields the number of elements as a `long` value.
`stream.findFirst()` `stream.findAny()`	Yields the first, or an arbitrary element as an `Optional<T>` — see Section 19.6.
`stream.max(`*comparator*`)` `stream.min(`*comparator*`)`	Yields the largest or smallest element as an `Optional<T>` — see Section 19.7.

`pstream.sum()`
`pstream.average()`
`pstream.max()`
`pstream.min()`
	The sum, average, maximum, or minimum of a primitive-type stream — see Section 19.8.

`stream.allMatch(`*condition*`)`
`stream.anyMatch(`*condition*`)`
`stream.noneMatch(`*condition*`)`
	Yields a `boolean` variable indicating whether all, any, or no elements match the condition — see Section 19.7.

`stream.forEach(`*action*`)`	Carries out the action on all stream elements — see Section 19.7.

Lec 8 Regular Expressions 正则表达式

Regular expression (or in short regex or RE) is a very useful tool that is used to describe a search pattern for matching the text.

Regex is nothing but a sequence of some characters that defines a search pattern.

Regex is used for parsing, filtering, validating, and extracting meaningful information from large text, such as logs and output generated from other programs.

regular expression	matches	does not match
.*spb.* *contains the trigraph spb*	raspberry crispbread	subspace subspecies
a* | (a*ba*ba*ba*)* *multiple of three b's*	bbb aaa bbbaababbaa	b bb baabbbaa
.*0.... *fifth to last digit is 0*	1000234 98701234	111111111 403982772
.*gcg(cgg|agg)*ctg.* *fragile X syndrome pattern*	...gcgctg... ...gcgcggctg... ...gcgcggaggctg...	gcgcgg cggcggcggctg gcgcaggctg

.* 不限数量

... 三个

.*0... 倒数第四是0

(..|..) 或

operation	example RE	matches	does not match
one or more	a(bc)+de	abcde abcbcde	ade bcde
character class	[A-Za-z][a-z]*	lowercase Capitalized	camelCase 4illegal
exactly j	[0-9]{5}-[0-9]{4}	08540-1321 19072-5541	111111111 166-54-1111
between j and k	a.{2,4}b	abcb abcbcb	ab aaaaaab
negation	[^aeiou]{6}	rhythm	decade
whitespace	\s	*any whitespace char (space, tab, newline...)*	*every other character*

Note. These operations are all *shorthand*.
They are very useful but not essential.

RE: (a|b|c|d|e)(a|b|c|d|e)*
shorthand: (a-e)+

. 任意　　$ 最后 ("here$" ⇒ "I like here")　　? 可有可无 (?)

^ 开头 ("^LO" ⇒ "LO IS OK")

* 指重复前一个0~多次　"10*" → "1, 10, 100, 1000"

[ ] 用[ ], [cn-cm], [^cn-cm]

"1[123]512" ⇒ "11512" "12512" "13512"

"[b-f]a" ⇒ "ba" "ca" "da" "ea" "fa"

"[^b-y]a" ⇒ "aa, za" 以外的区域！

+ 指重复前一个1~多次 "10+" → "10, 100, 1000"

$m^+ = mm^*$

^需加才可保持原义. [^1-8]00 ⇒ "900"

选择 | "^aba|b$" ⇒ aba, abb

重复 \{n\} {n} 前面的表达式要匹配n次

{n,} \{n,\} —— 最多匹配n次

{n,} \{n,\} —— 最少匹配n次

{n,m} \{n,m\} —— 匹配n-m次

\b

引用: String a1 = "...."
a1.matches("+-%") ;
boolean

```java
import java.util.List;
import java.util.stream.*;
import java.util.regex.*;

public class AsPredicateExample {
   public static void main (String[] args) {
      final String[] monthsArr =
         { "10", "0", "05", "09", "12", "15", "00", "-1", "100" };
      final Pattern validMonthPattern =
         Pattern.compile( "^(?:0?[1-9]|1[0-2])$" );
      List<String> filteredMonths =
         Stream.of( monthsArr )
               .filter( validMonthPattern.asPredicate() )
               .collect( Collectors.toList() );
      System.out.println( filteredMonths );
   }
}
```

一个调用

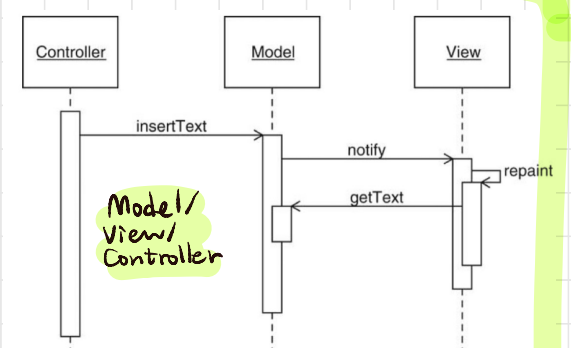
```
H:\work\CS209A_19S\notes08>javac AsPredicateExample.java

H:\work\CS209A_19S\notes08>java AsPredicateExample
[10, 05, 09, 12]
```

Lec9 Program Qualities Evaluation:

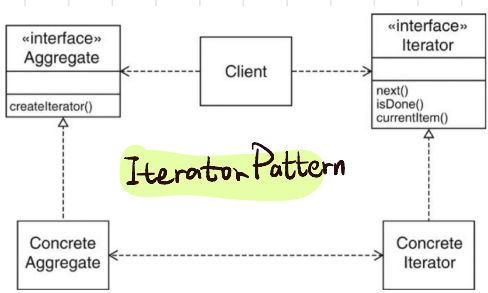
Correctness: Robustness: Efficiency:

Lec10 Design Pattern

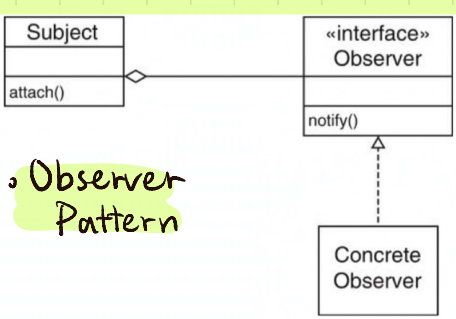
- Short Passages Pattern



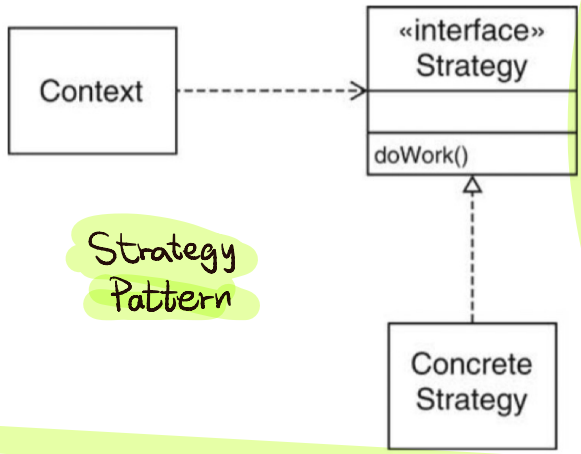
Model/ View/ Controller



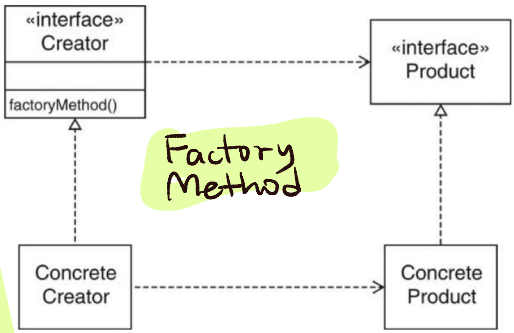
Iterator Pattern



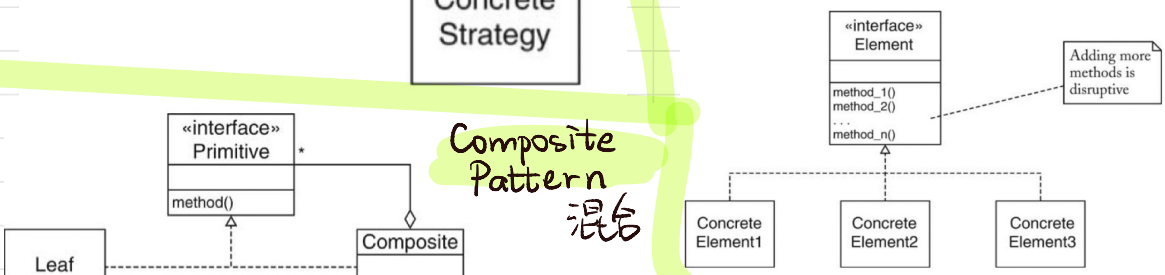
- Observer Pattern



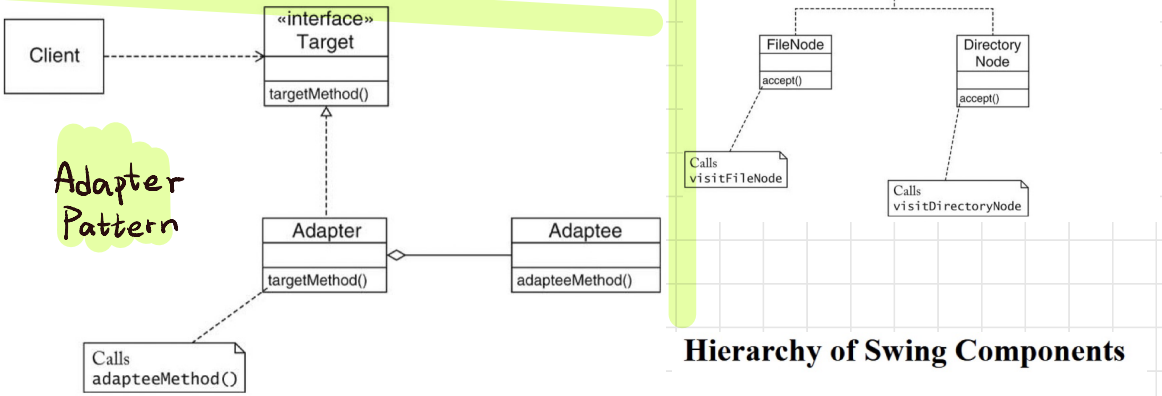
Strategy Pattern



Factory Method



Composite Pattern 混合



Visitor

Adapter Pattern

Hierarchy of Swing Components

- History: First came AWT, Abstract Window Toolkit
- Used *native* components
- Subtle platform inconsistencies
- Write once, run anywhere ->
  Write once, debug everywhere
- Swing paints components onto blank windows
- Supports multiple *look and feel* implementations

Lec11 CarMover, SceneEditor ?

Lec12、13不懂怎么整

Lec12

- Threads in multithreaded processors may or may not be same as software threads
- Process:
  - An instance of program running on computer
- Thread: dispatchable unit of work within process
  - Includes processor context (which includes the program counter and stack pointer) and data area for stack
  - Thread executes sequentially
  - Interruptible: processor can turn to another thread
- Thread switch
  - Switching processor between threads within same process
  - Typically less costly than process switch

Client
Sever
connection

Table 1 HTTP Commands

Command	Meaning
GET	Return the requested item
HEAD	Request only the header information of an item
OPTIONS	Request communications options of an item
POST	Supply input to a server-side command and return the result
PUT	Store an item on the server
DELETE	Delete an item on the server
TRACE	Trace server communication

Lec14 Name, Scope, Binding, FreeMarker, MyBatis, JSON, Gson

- A name is exactly what you think it is
  - Usually think of identifiers but can be more general
  - symbols (like '+' or '_') can also be names

- A binding is an association between two things, such as a name and the thing it names

两事物的关联

- The scope of a binding is the part of the program (textually) in which the binding is active

范围

Binding Everywhere

- Domain name ➔ IP Address ➔ NIC with MAC
- Variable ➔ value or reference (pointer)
- Polymorphism: Late/Dynamic Binding by Method looking up

```
SuperType v = new SubClass();    // java code
v.methodA();
```

SuperType could be a class or an interface.

Compiling time just check methodA() declaration from SuperType.

Runtime look up definition of methodA() starting from SubClass upwards.

Binding Time is the point at which a binding is created

- language design time
  - program structure, possible type
- language implementation time
  - I/O, arithmetic overflow, stack size, type equality (if unspecified in design)
- program writing time
  - algorithms, names
- compile time
  - plan for data layout
- link time
  - layout of whole program in memory
- load time
  - choice of physical addresses

Implementation decisions (continued):

- run time
  - value/variable bindings, sizes of strings
  - subsumes
    - program start-up time
    - module entry time
    - elaboration time (point a which a declaration is first "seen")
    - procedure entry time
    - block entry time
    - statement execution time

Life Time Storage

- Storage Allocation mechanisms – Static
  - Stack
  - Heap

- Static allocation for – code
  - globals
  - static or own variables
  - explicit constants

(including strings, sets, etc) – scalars may be stored in the instructions

Scope Rules

- A *scope* is a program section of maximal size in which no bindings change, or at least in which no re-declarations are permitted
- In most languages with subroutines, we OPEN a new scope on subroutine entry:
  - create bindings for new local variables,
  - deactivate bindings for global variables that are re-declared (these variable are said to have a "hole" in their scope)
  - make references to variables

Scope Rules

- On subroutine exit:
  - destroy bindings for local variables
  - reactivate bindings for global variables that were deactivated
- Algol 68:
  - ELABORATION = process of creating bindings when entering a scope
- Ada (re-popularized the term elaboration):
  - storage may be allocated, tasks started, even exceptions propagated as a result of the elaboration of declarations

- With STATIC (LEXICAL) SCOPE RULES, a scope is defined in terms of the physical (lexical) structure of the program
  - The determination of scopes can be made by the compiler
  - All bindings for identifiers can be resolved by examining the program
  - Typically, we choose the most recent, active binding made at compile time
  - Most compiled languages, C++ and Java included, employ static scope rules

- The classical example of static scope rules is the most closely nested rule used in block structured languages such as Algol 60 and Pascal
  - An identifier is known in the scope in which it is declared and in each enclosed scope, unless it is re-declared in an enclosed scope
  - To resolve a reference to an identifier, we examine the local scope and statically enclosing scopes until a binding is found

- Note that the bindings created in a subroutine are destroyed at subroutine exit
- Obvious consequence when you understand how stack frames are allocated and deallocated
- The modules of Modula, Ada, etc., give you closed scopes without the limited lifetime
  - Bindings to variables declared in a module are inactive outside the module, not destroyed
  - The same sort of effect can be achieved in many languages with *own* (Algol term) or *static* (C term) variables

- Access to non-local variables STATIC LINKS
  - Each frame points to the frame of the (correct instance of) the routine inside which it was declared
  - In the absence of formal subroutines, *correct* means closest to the top of the stack
  - You access a variable in a scope k levels out by following k static links and then using the known offset within the frame thus found

- The key idea in **static scope rules** is that bindings are defined by the physical (lexical) structure of the program.
- With **dynamic scope rules**, bindings depend on the current state of program execution
  - They cannot always be resolved by examining the program because they are dependent on calling sequences
  - To resolve a reference, we use the most recent, active binding made at run time
- Dynamic scope rules are usually encountered in interpreted languages
  - early LISP dialects assumed dynamic scope rules.
- Such languages do not normally have type checking at compile time because type determination isn't always possible when dynamic scope rules are in effect

- Since most large programs are constructed and tested incrementally, and some programs can be very large, languages must support separate compilation
- Compilation units usually a “module”
  - Class in Java/C++
  - Namespace in C++ can link separate classes
  - More arbitrary in C

Conclusions

- The morals of the story:
  - language features can be surprisingly subtle
  - Determining how issues like binding, naming, and memory are used have a huge impact on the design of the language
  - Static vs. dynamic scoping is interesting, but all modern languages use static scoping
  - most of the languages that are easy to understand are easy to compile, and vice versa

故事的寓意：

- 语言功能可能会出人意料地微妙
- 确定如何使用绑定、命名和内存等问题会对语言的设计产生巨大影响
- 静态和动态范围比较有趣，但所有现代语言都使用静态范围
- 大多数易于理解的语言都易于编译，反之亦然

Freemarker: A template engine

Mybatis: A first class persistence framework with SQL!

JSON: is a lightweight data-interchange format.
Notation Built on two structures: a collection of name/value pairs
a ordered list of values.

Gson: a Java library that can be used to convert Java Objects into their JSON representation.

√ 23. If a Java interface defines two or more methods you cannot use lambda expressions with it.

X 24. In a Graphical User Interface, you can only store a container in a different type of container (for instance you can only add a vertical box to a grid or a horizontal box but not another vertical box).

CS209A-f21 Quiz on Dec. 14

1. A and E are classes, B and D are interfaces. Which of the following statements are correct?

   a) `interface F implements B, D { }`
   b) `interface F implements D { }`
   c) `interface F extends D { }`
   d) `interface F extends E { }`
   e) `interface F extends B, D { }`

接口可多继承接口
类只接一个接口

2. What is polymorphism?

   a) Polymorphism is a technique to define different objects of the same type
   b) Polymorphism is the ability of an object to take on many forms
   c) Polymorphism is a technique to define different methods of the same type
   d) None of the above

多态

Poly = several, morph = shape (in Greek)

3. When we refer to a "member of a class" we mean:

   a) An attribute
   b) A method
   c) An attribute or method
   d) A sub-class

类成员

"member" is a generic term for whatever belongs to a larger group (group member, family member ...). Anything inside a class is a "member".

4. If classes Student, Staff, and Faculty extend the class Person, which of the following make sense?

   a) `Faculty[] faculties = { new Person(), new Staff(), new Student() };`
   b) `Staff[] staff = { new Person(), new Faculty(), new Student() };`
   c) `Person[] persons = { new Faculty(), new Staff(), new Student() };`

5. What is the output of the following program?

   a) `0 0 0`
   b) `0 2 1`
   c) `0 1 0`
   d) `Compile error`

The function actually counts how many '8' there are in the

```java
class Recursion {
    int func(int n) {
        int result = 0;
        if (n < 10) {
            if (n == 8) {
                result = 1;
            }
        } else {
            result = func(n % 10) + func(n / 10);
        }
        return result;
    }
}

public class Prog {
  public static void main(String args[]) {
       Recursion obj = new Recursion() ;
       System.out.print(obj.func(123) + " ");
       System.out.print(obj.func(8328) + " ");
       System.out.println(obj.func(8325));
  }
}
```

6. What is the output of the following program?

a) false true
b) true false
c) true false
d) false false

```
interface Int{}
class Simple implements Int{}
class SimpleTest {
   public static void main(String[] args) {
     try {
       Class c=Class.forName("Simple");
       System.out.print(c.isInterface() + " ");
       c=Class.forName("Int");
       System.out.println(c.isInterface());
     } catch (Exception e) {
       System.out.println(e);
     }
   }
}
```

7. "static" key word in java indicates:

a) Something that you cannot change
b) Something that is attached to the class, not to a particular object instance
c) Something that cannot throw exceptions
d) Something that cannot be overridden (methods) or extended

8. The protocol that takes a message from a network to another network is:

a) FTP
b) TCP
c) IP
d) HTTP

IP = Internet Protocol.
Internet means to net(work)s what *international* means to nations.

FTP = File Transfer Protocol (more like HTTP, but dedicated to exchanging files)

TCP = Transmission Control Protocol, computer-to-computer (uses IP underneath)

HTTP = Hyper Text Transfer Protocol, you know this one.

。没有 try /catch 就不能用 finally

```
public class Test {

    public static void main(String[] args) {
        System.out.println("return value of getValue(): "
                           + getValue());
    }

    public static int getValue() {
        try {
            return 0;
        } finally {
            return 1;
        }
    }
}
```

return value of getValue(): 1

9. In the program below where will the references to the variables a, b, and c, be stored?

a) Heap, heap, heap
b) Heap, stack, heap
c) Heap, stack, stack
d) Heap, heap, stack

```
class A {
    private String a = "aa";

    public boolean methodB() {
        String b = "bb";
        final String c = "cc";
    }
}
```

"aa", "bb" and "cc" are all in the heap but we are talking about references to these values.

Local variables are variables defined inside methods, constructors or blocks

Instance variables are variables within a class but outside any method

20. Which of the following steps are performed when accessing a database via JDBC?

a) Loading the JDBC driver.
b) Setting up the connection.
c) Executing queries or applying changes to the database
d) Close the connection

12. Which of the following belongs to meta-annotations?

a) @Override
b) @Retention
c) @Deprecated
d) @SuppressWarnings()

A meta-annotation is an annotation about annotation. The other annotations refer to the annotated code.

13. Which of the following are valid retention policy types in Java (multiple answers)?

保留策略类

a) SOURCE
b) CLASS
c) RUNTIME

FYI: The @Retention annotation indicates how long annotations with the annotated type are to be retained.

Can be one of:
- CLASS: annotations are to be recorded in the class file by the compiler but need not be retained by the VM at run time.
- RUNTIME: Annotations are to be recorded in the class file by the compiler and retained by the VM at run time, so they may be read reflectively.
- SOURCE: Annotations are to be discarded by the compiler.

17. Which of the following can obtain the location of file Reflection.class

```
Public class Reflection { }
```

a) Reflection.class.getClassLoader().getResource("Reflection.class")
b) Reflection.getClass().getResource("Reflection.class")
c) Reflection.getClass().getClassLoader().getResource("Reflection.class")
d) Reflection.class.getClassLoader("Reflection.class")

getClass() applies to an object, .class to a class. File "Reflection.class" is a resource for the JVM.

19. What is the main purpose of JDBC?

a) Create a connection to the database.
b) Send SQL statements to the database.
c) Processing data and query results
d) All of the above

A TableView is used to display data retrieved from a data source (often a database, but it could as well be the result of streaming an in-memory collection, or data retrieved from the network). You rarely know how many rows you'll display, and the wisest choice is a ScrollPane.

Static: Something that is attached to the class, not to a particular object instance

- Ant、Make、Maven are build tools, Junit is Testing tool.

27. The time taken by a message to travel between computers is called:

a) Bandwidth
b) Routing
c) Latency
d) Delay

Bandwidth refers to how much data can be sent at the same time, not to speed. Routing is finding a circuit (route) to go to computer to computer; it's about traffic redirection.
Delay is just a general term.

11. The following code:

```
public class Foo {
   int year = 2019;
   public static void main (String [] args) {
      System.out.println( "Hello " + year );
   }
}
```

编译错误：static int

- Garbage Collection can't guarantees that a program will not run out of memory.
- Every "try" must followed by a catch or finally.

8. **For the difference between ArrayLists and LinkedList, which is correct? (Multiple answers)**

a. The ArrayList data structure is implemented by a dynamic array, the LinkedList data structure is based on nodes holding references.
b. For random access (get and set), an ArrayList performs better than a LinkedList because a LinkedList needs to follow references.
c. For add and remove operations, a LinkedList is preferred because the ArrayList needs to move data.

To make an object serializable:

a. You don't need to do anything - all objects are serializable by default.
b. You only need to say that it implements the Serializable interface.
c. You need to say that it implements the Serializable interface and implement the two methods writeObject() and readObject() defined in the interface.

It's not done by default because it adds overhead (more code) that not everybody needs.

If you serialize an object that contains references, when you reload it the referenced objects will go at exactly the same place in memory:

a. True
b. False

Memory location is the business of the operating system, not yours. It can put things wherever it likes, as long as it returns references to you.

2. Which of the following statements about Java Threads is correct?
   A. Java threads don't allow parts of a program to be executed in parallel
   B. Java is a single-threaded language
   C. Java's garbage collector runs as a high priority thread
   D. Ready, running and sleeping are 3 states that a thread can be in during its life cycle
   E. Every java application is not multithreaded.

Exercise 7.3. Which of the following types are subtypes of another type?

(a) `Object`
(b) `int`
(c) `long`
(d) `int[]`
(e) `Object[]`
(f) `int[][]`
(g) `Rectangle`
(h) `Rectangle[]`
(i) `Rectangle2D[]`
(j) `Cloneable`

Note that each is a subtype of itself by rule 1, but we ignore that possibility.

a. Object: not a subtype
b. int: not a subtype
c. long: not a subtype
d. int[]: subtype of Object by rule 6, of Cloneable by rule 7
e. Object[]: subtype of Object by rule 6, of Cloneable by rule 7
f. int[][]: subtype of Object[] by rule 5, of Object by rule 6, of Cloneable by rule 7
g. Rectangle: subtype of Cloneable by rule 3, of Object by rule 6
h. Rectangle[]: subtype of Rectangle2D[] and Object[] by rule 5, of Object by rule 6, of Cloneable by rule 7
i. Rectangle2D[]: subtype of Object[] by rule 5, of Object by rule 6, of Cloneable by rule 7
j. Cloneable: subtype of Object by rule 6

12. If class Arc extends class Shape, which one of the following code segment is correct:

A. `Arc x = new Arc(); Object y = (Object)x; Arc z = y;`
B. `Arc x = new Arc(); Shape y = x; Arc z = (Arc)y;`
C. `Shape y = new Shape(); Arc x = (Arc)y; Shape z = x;`
D. `Shape y = new Shape(); Arc x = (Arc)y; Shape z = (Shape)x;`

从子类向父类的转换不需要什么限制，只需直接将子类实例赋值给父类变量即可

父类变量向子类转换必须通过显式强制类型转换

1. 定义A为父类，B、C为A的子类，以下

```
1 A a = new B();
2 A c = (A)new C();
```

正确，但a,c都只能调用A父类中已存在的方法，

```
1 B b =  (B) new A();
```

错误，父类无法转换为子类

```
1 B b = (C)new C();
```

错误，C类无法转换为B类，二者没有实际关系