Review

Lecture1

Information Systems

Computing takes place information systems

Software

The programs that run on a computer with data and documents are referred to as software

---

Program

A sequence of instructions that specifies how to perform a computation

---

Instruction

- input: get data
- output: output data
- math: perform math operations
- testing: check for conditions, run statements
- repetition: perform actions

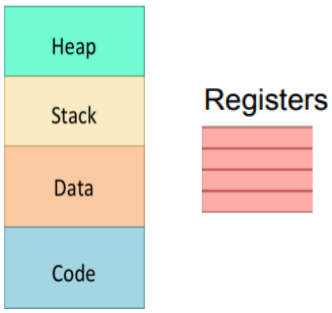
The Fetch Execute Cycle

(first proposed by John von Neumann)

- Fetch: gets the next program command from the computer's memory
- Decode: deciphers what the program is telling the computer to do
- Execute: carries out the requested action
- Store: saves the results to a Register or Memory

Von Neumann Architecture

computer memory is split between a part that contains



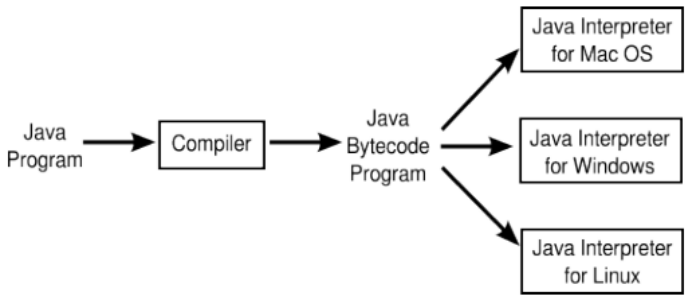
- code : executable instructions
- static data
- stack : volatile data associated with functions come and go
- heap : objects are created

The processor also contains registers : data and instructions are brought from memory for Registers processing

---

Java Virtual Machine (JVM)

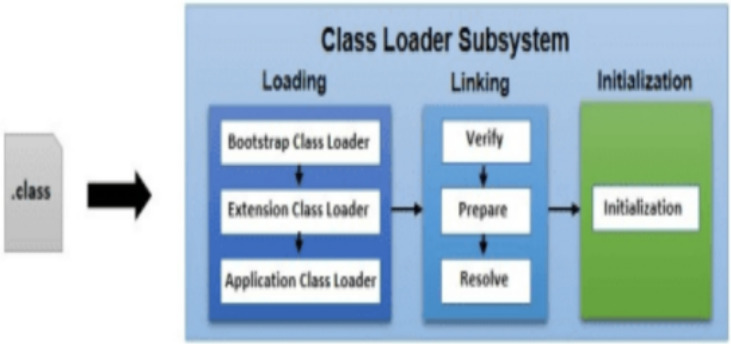
Original Translate: High-level programming languages –> (translation by complier) –> Machine language that can only be run on one type of computer (each type of computer has its own individual machine language)



Compilation: `javac xxx.java` turn a `.java` file to a `.class` file (byte code program)

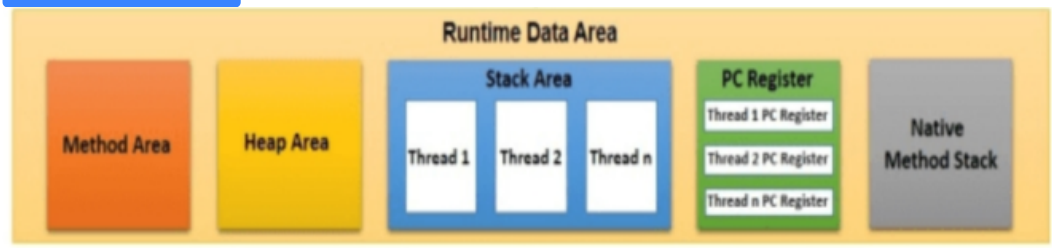
Execution: `java xxx (args)`

- Interpreter (执行 Compile 过后的 .class 文件)
  - class loader

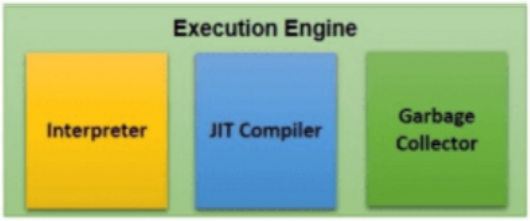


    - **loading the .class files**(sometimes grouped together into a .jar, Java ARchive)
    - **locate every single class** referenced in the code
    - then to set up everything so that when you call a **method** the engine knows where to find the instructions to execute

        - initialize everything and you'll be able to call **main()**
    - Runtime Data Areas



        - find all the data and instructions
    - Execution Engine



        - **Interpreter**: converts the "**run everywhere**" code found in the .class to some "**run on this particular hardware**" instructions that the processor can execute (Interpreting code is slow)
        - **Just In Time Compiler(JLT)**: that converts but then reuses the conversion when the same code is executed again
        - **Garbage Collector**: identifies objects that are no longer in use and reclaims the memory they are using

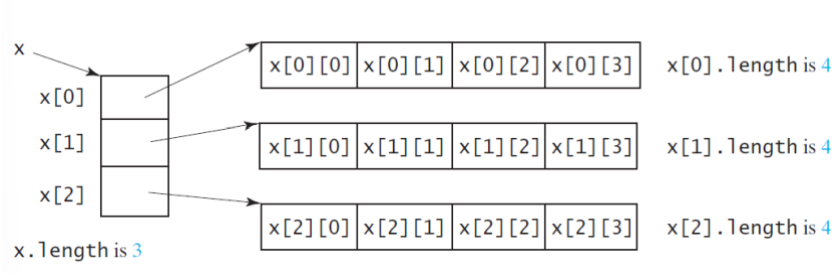
---

Array

Java reserves sufficient space in memory to store the specified number of elements. This process is called memory allocation.

```
int[] a = new int[7];
double[][] a = new double[m][n];
double[][][] a = new double[i][j][k];
```

```
double[][] a = {
    { 99.0, 85.0, 98.0, 0.0 },
    { 98.0, 57.0, 79.0, 0.0 },
    { 92.0, 77.0, 74.0, 0.0 },
    { 94.0, 62.0, 81.0, 0.0 },
    { 99.0, 94.0, 92.0, 0.0 },
    { 80.0, 76.5, 67.0, 0.0 },
    { 76.0, 58.5, 90.5, 0.0 },
    { 92.0, 66.0, 91.0, 0.0 },
    { 97.0, 70.5, 66.5, 0.0 },
    { 89.0, 89.5, 81.0, 0.0 },
    {  0.0,  0.0,  0.0, 0.0 }
};
```

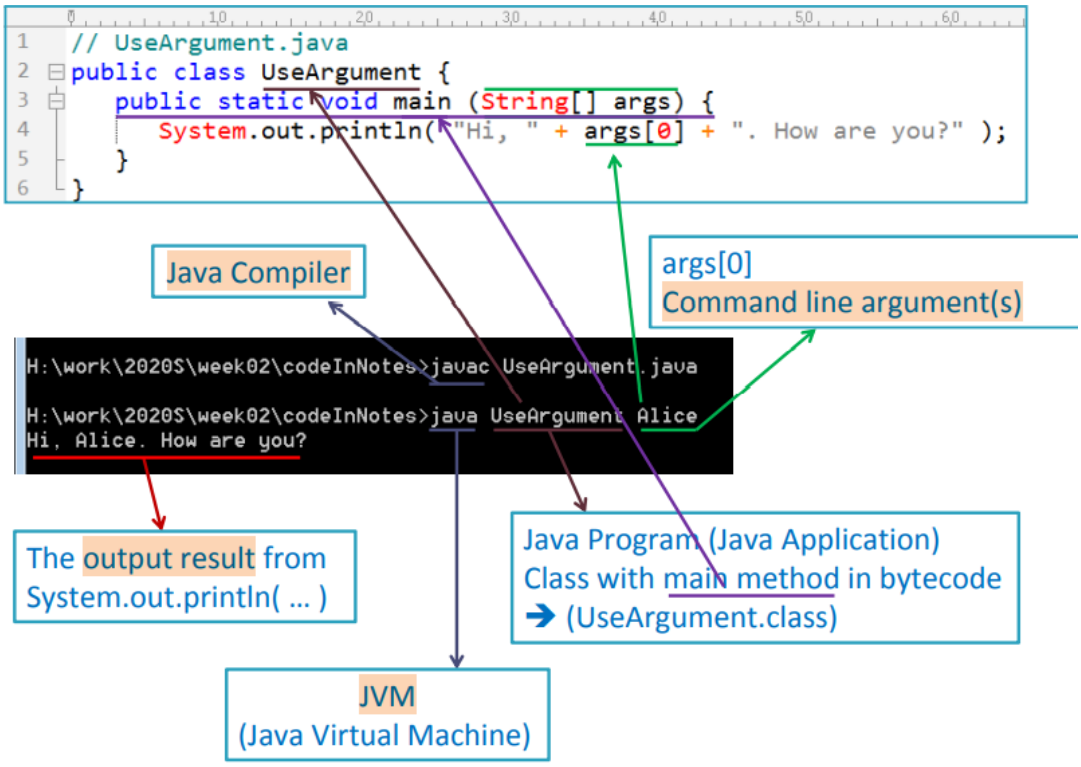


Ragged Array

There is **no requirement** that all rows in a two-dimensional array **have the same length**—an array with rows of nonuniform length is known as a ragged array.

```
int[][] raggedArray = {
   { 1, 2, 3, 4, 5 },
   { 2, 3, 4, 5 },
   { 3, 4, 5 },
   { 4, 5 },
   { 5 }
};
```

Command Line



Lecture2

Complex & Complicated

Complex (错综复杂的) ≠ complicated (复杂难懂的)

- Complex : composed of many simple parts related to one another
- Complicated : not well understood, or explained

Function-Behavior-Structure

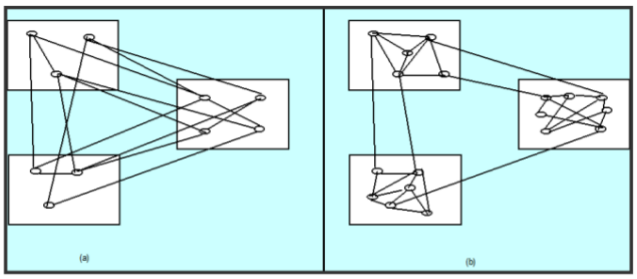
Function is connected with behavior, and behavior is connected with structure.

There is no connection between function and structure!

- Function(F) : the teleology (purpose) of the design object
- Behavior(B) : the attributes that can be derived from the design object's structure
- Structure (S) : the components of the design object and their relationships

How to Decompose Systems into Modules

高内聚，低耦合，信息隐藏



high coupling
(low cohesion)

low coupling
(high cohesion)

- High Cohesion within modules
- Loose Coupling between modules
- Information Hiding

modules : A complex system may be divided into simpler pieces

modular (模块化的): A system that is composed of modules

Soc : Supports application of separation of concerns(Soc 关注点分离)

- when dealing with a module we can ignore details of other modules

---

How to Manage Changes in developing process

- minimizing the amount of code that must change
- it easy to work out which code must change
- having the code that must change live together

Criteria for Good Programs

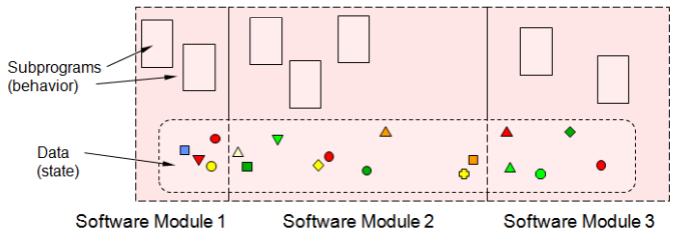
- Run Effectively (Correctly) & Efficiently (Objectives 目的)
- Easy to be Extended and Modified (Approaches 手段)
- Easy to be Understood (Pre-conditions 前提)

Benefit of encapsulation

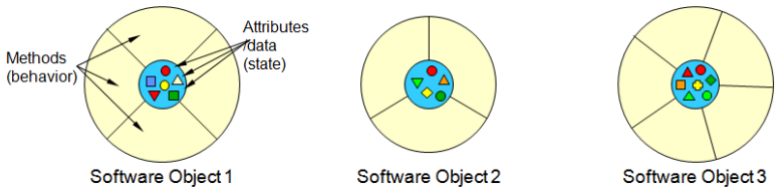
- loose coupling
- protected variation
- exporting an API

Module vs Object

Modules are loose groupings of subprograms and data



Objects encapsulate data



OOP

Object Oriented programming (OOP)

- Things in the world know things: instance variables
- Things in the world do things: methods

Objects have state and behavior

Access modifiers: control the visibility of your code to other programs.

public: member is accessible whenever the class is accessible.

private: member is only accessible within the class.

default: member is only accessible within package.

protected: member is only accessible within package and extended classes.

Accessor methods: just read the data

Mutator methods: modify the data

一个 java 文件中只能定义 一个 public class，可以定义 多个非 public class 的

In Java, method calls are always determined by the type of the actual object, not the type of the variable containing the object reference. This is called Dynamic Method Lookup / Dynamic Binding

Dynamic method lookup allows us to treat objects of different classes in a uniform way. This feature is called polymorphism

```
public class MySubClass extends TheSuperClass {

   public MySubClass () { // 如不定义无参数构方法，JVM会自动生成一个无参数构造方法
      super(); // 如果无明确调用superClass的构造方法，会自动调用超类无参数构造方法
      ...
   }

   // 可定义自己的方法
   // 也可重写(override)superClass的方法
   // 可定义自己的instance varibales
   // 但不能访问superClass任何private声明的变量和方法
}

TheSuperClass v1 = new TheSuperClass;   // ok
TheSuperClass v2 = new MySubClass();    // ok
MySubClass v3 = new MySubClass();       // ok

v1 = v3;  // ok, Type Promotion
v3 = v2;  // wrong!
v3 = (MySubClass) v2;   // ok , Type Casting
v3 = (MySubClass) v1;   // wrong!
```

```
public abstract class AnAbstractClass {
   // No Constructor
   public abstract SomeType abstractMethodA (Type1 v1...);  // 抽象方法只声明，无定义
   public SomeType methodB (Typex vx...) { ... }  // 非抽象方法有声明有定义
   ...
}

public class MyClass extends AnAbstractClass {

   public SomeType abstractMethodA (Type1 v1...) { ... }  // 定义超类中的抽象方法
   public SomeType methodB (Typex vx...) { ... }  // 重写 (override) 超类的方法
                                                  // 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
}

AnAbstractClass v1 = new AnAbstractClass();  // wrong, 抽象类不能实例化
AnAbstractClass v2 = new MyClass();   // ok, 可实现多态
MyClass v3 = new MyClass();  // ok
```

```
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
```

- 接口可以定义常量，所有定义的成员变量都会自动加上 `public static final` 修饰
- 抽象类里可以定义变量和方法

```
1  public interface test{
2      int a = 10; // public static final int a = 10;
3      public static final int b = 10;
4      // a 和 b 的性质相等
5      // 接口的变量是不可变的
6  }
```

- 接口可以多继承接口：Interface1 extends Interface2, Interface3, Interface4
- 抽象类可以实现接口：抽象类实现接口时，可以完全重写或覆盖接口中的方法，也可只重写接口中的部分方法
- 抽象类可以继承实体类：抽象类可以继承实体类，但前提是实体类必须有明确的构造函数
- outer class 不可继承自多个具体 class，可在其内部设多个 inner class，每个 inner class 都能各自继承某一实现类，inner class 不受限于 outer class 是否已经继承自某一实现类

Exception

There are two aspects to dealing with program errors: detection and handling

Exceptions are (special) Objects

```
1  throw new MyException();
```

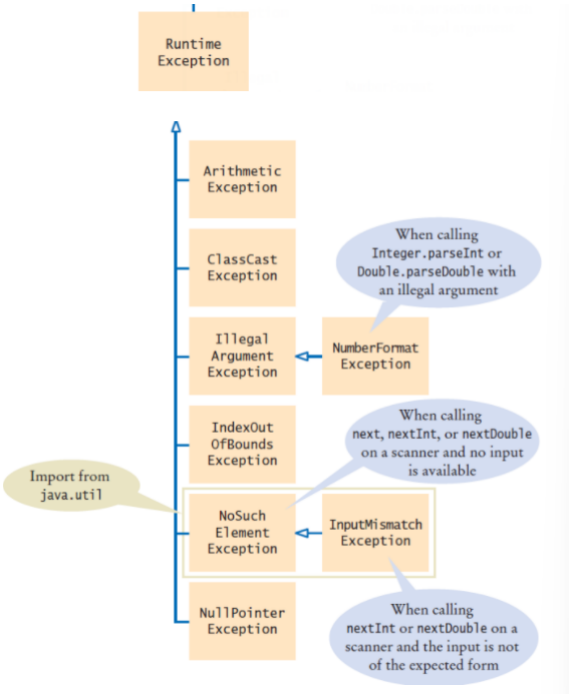
- You throw an object, not a type
- As an exception is an object, is must be instantiated (created from the template that a class defines as a type) using new when you need it and throw it

Exception vs Error

- Exception handling provides a way to handle and recover from problems or a way to quit the program in a graceful way
- Errors represent serious represent serious unrecoverable problems

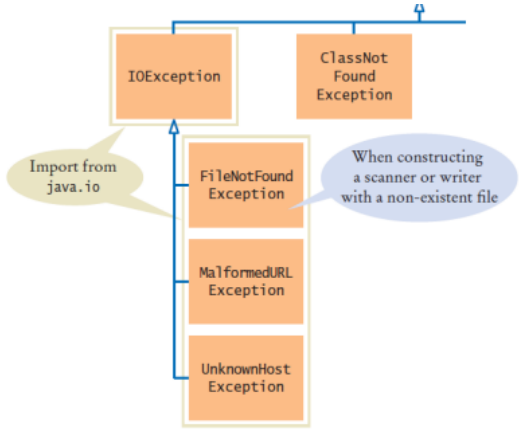
Exception Type

unchecked exception



- RuntimeException 的子类
- Java 不强制检查 unchecked exception
- 自定义异常继承 RuntimeException 将不要求异常处理

checked exception



- 除了 unchecked exception 和 error 之外的所有
- Java 强制检查 checked exception
    - 要么方法声明 throws
    - 要么使用 try-catch 等进行处理
- 自定义异常继承 Exception/IOEException 将必须要求异常处理

```
class MyException extends Exception {
    public MyException() {}
    public MyException(String msg) {
        super(msg);
    }
}
```

```
class ExceptionTest {

    public void testCatch(){
        try {
            testThrows();
        } catch (MyException e) {
            e.printStackTrace();
        }
    }

    public void testThrows() throws MyException {
        test();
    }

    public void test() throws MyException {
        throw new MyException("This is MyException");
    }
}
```

```
public void test() throws MyException, IOException {
    throw new IOException();
}
```

Try-with Resource

More generally, you can declare variables of any class that implements the AutoCloseable interface in a try-with-resources statement.

Try/finally

The try/finally statement is rarely required because most Java library classes that require cleanup implement the AutoCloseable interface.

Internationalization

- UTF-32 is an encoding scheme for Unicode that employs four bytes to represent every code point
  - For example: "T" in "UTF-32" is "00000000 00000000 00000000 01010100" (the first 3 bytes are unnecessary)
- UTF-8 is another encoding scheme for Unicode which employs a variable length of bytes to encode
  - For example: "T" in "UTF-8" is "01010100"
  - "語" in "UTF-8" is "11101000 10101010 10011110"

File IO

If you need to process files with accented characters, Chinese characters, or special symbols, you should specifically request the UTF-8 encoding.

Lecture3

Data Type

- Data Type : a set of values and a set of operations on those values
- Abstract Data Type(ADT) : a data type whose representation is hidden from the client
    - Clients can use ADTs without knowing implementation details

data type	*set of values*	*examples of operations*
Color	three 8-bit integers	get red component, brighten
Picture	2D array of colors	get/set color of pixel (i, j)
String	sequence of characters	length, substring, compare

> To use a data type, you need to know:
> - Its name (capitalized, in Java).
> - How to *construct* new objects.
> - How to *apply operations* to a given object.

> To construct a new object
> - Use the keyword new to invoke a *constructor*.
> - Use data type name to specify type of object.

> To apply an operation (invoke a method)
> - Use object name to specify which object.
> - Use the dot operator to indicate that an operation is to be applied.
> - Use a method name to specify which operation

declare a variable (object name)

call a constructor to create an object

```
String s;
s = new String( "Hello, World!" );
System.out.println( s.substring( 0, 5 ) );
```

object name

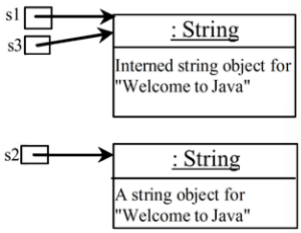
call an instance method that operates on the object's value

```
String s1 = "Welcome to Java";

String s2 = new String("Welcome to Java");

String s3 = "Welcome to Java";

System.out.println("s1 == s2 is " + (s1 == s2));
System.out.println("s1 == s3 is " + (s1 == s3));
```



- s1 == s2 is false
- s1 == s3 is true

Create Data Type

To create a data type, you need provide code that
- Defines the set of values (instance variables).
- Implements operations on those values (methods).
- Creates and initialize new objects (constructors).

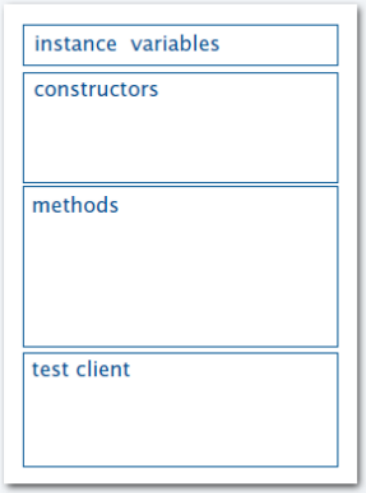
In Java, a data-type implementation is known as a *class*.

Instance variables
- Declarations associate variable names with types.
- Set of type values is "set of values".

Methods
- Like static methods.
- Can refer to instance variables.

Constructors
- Like a method with the same name as the type.
- No return type declaration.
- Invoked by new, returns object of the type.

A Java class



Lecture4

Bitwise Operators

Where b_n is a sign bit.

```
byte :     8 bits  (1 byte),  n = 7
short:     16 bits (2 bytes), n = 15
int:       32 bits (4 bytes), n = 31
long:      64 bits (8 bytes), n = 63
```

```
// x=A and y=B   ➔ Pre-Condition
x = x ^ y;
// x=A^B and y=B
y = x ^ y;
// x=A^B and y=A
x = x ^ y;
// x=B and y=A   ➔ Post-Condition
```

```
&& and || follow short-circuit break rule,
but & and | calculate all sub-terms.

if (x != 0 && y/x > 5)   √
   // y/x > 5 will not calculated when x is 0
if (x != 0 & y/x > 5)    // when x is 0, y/x > 5 will cause:
Throws java.lang.ArithmeticException: / by zero
```

XOR

a	b	a⊕b
0	0	0
0	1	1
1	0	1
1	1	0

```
Both  1 ^ 1 and 0 ^ 0 get 0, and
      0 ^ 1 and 1 ^ 0 get 1.

For any integer a and b:
      a ^ b ^ b get a.  // b^b get 0, a^0 get a.
```

Collections

In a *TreeMap*, the key/value associations are stored in a sorted tree, in which they are sorted according to their `keys`. For this to work, it must be possible to compare the keys to one another.

This means either that the keys must implement the interface *Comparable<K>*, or that a *Comparator* must be provided for comparing keys.
(The *Comparator* can be provided as a parameter to the *TreeMap* constructor.)

Note that in a *TreeMap*, as in a *TreeSet*, the `compareTo()` (or `compare()`) method is used to decide whether two keys are to be considered the same.

If `map` is a variable of type *Map<K,V>*, then
The value returned by `map.keySet()` is a "view" of keys in the map
implements the *Set<K>* interface

One of the things that you can do with a *Set* is get an *Iterator* for it
and use the iterator to visit each of the elements of the set in turn.

`map.values()` returns an object of type *Collection<V>* that contains
all the values from the associations that are stored in the map.
The return value is a *Collection* rather than a *Set* because it can
contain duplicate elements.

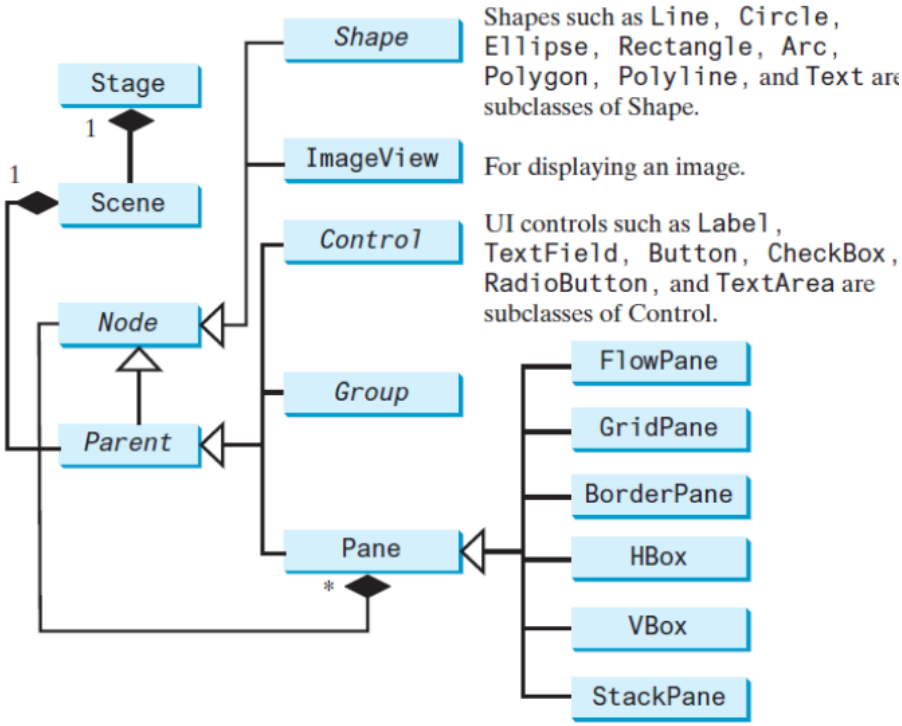
`map.entrySet()` returns a set that contains
all the associations from the map.
The elements in the set are objects of type *Map.Entry<K,V>*.
The return type is written as *Set<Map.Entry<K,V>>*.

Each `Map.Entry` object contains one key/value pair, and defines
methods `getKey()` and `getValue()` for retrieving the key and the value.

Because the priority queue needs to be able to tell which element is the smallest,
the added elements should belong to a class that implements the `Comparable` interface.
Thus, each removal operation extracts the *minimum* element from the queue.

Lecture5

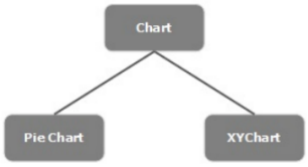
- Stage is a window for displaying a scene that contains nodes
- Multiple stage can be displayed in a JavaFX program
- In JavaFX, a scene object is a hierarchical data structure that **contains the contents of a window that is made up of "nodes" that represent visible components in the window**, such as buttons and text-input boxes.
- Nodes can be shapes, image views, UI controls, groups and panes



- The java coordinate system is measured in pixels, with (0, 0) at it's upper-left corner
- Group class is often used to group nodes and to perform transformation and scale as a group
- Resizable: Panes, UI control objects
- Not resizable: Group, Shape, Text
- The Image class represents a graphical image, and the ImageView class can be used to display an image
- Shape class is the abstract base class, it is a node, this class is the root of all shape class
- Polygon is closed and Polyline is not closed

Lecture6

- A chart is a graphical representation of data
- To create a chart, you need to
  - Define the axis of the chart
  - Instantiate the respective class
  - Prepare and pass data to the chart
- JavaFX provides support for carious PieCharts and XY Charts



  - XY-Chart include AreaChart, BarChart, BubbleChart, LineChart, ScatterChart, StackedAreaChart, StackedBarChart...
  - Chart is an abstract class
- JavaFX does not allow multiple plots of the different types to be displayed simultaneously on the same scene

- Saving a Plot to File

  It is critical to turn off animation via `aChart.setAnimated(false)` someplace after the chart is instantiated and before data is added to the chart with `aChart.getData.add()` or its equivalent:

```
/* do this right after the chart is instantiated */
scatterChart.setAnimated(false);
...
/* render the image */
stage.show();
...
/* save the chart to a file AFTER the stage is rendered */
WritableImage image = scatterChart.snapshot(new SnapshotParameters(), null);
File file = new File("chart.png");
ImageIO.write(SwingFXUtils.fromFXImage(image, null), "png", file);
```

Lecture7

- Streams: I/O Streams, byte/character streams
- Stream Processing: A functional Processing Style
  - Producing Stream
  - Transform Stream
  - Collecting Stream
- Once you obtain a result from a stream. the stream is "used up", and you can no longer apply other operations to it

Lecture8

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.

语法分析，过滤，检验，抽取有意义的信息

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

- Alphabet: a finite set of symbols
- String: a finite sequence of alphabet symbols
- Formal Language: a set of strings (possibly infinite) all over the same alphabet
- A regular expression(RE): a string of symbols that specifies a formal language, recursively, using the union, concatenation, and closure operations on sets
- A language is regular if and only if it can be specified by an RE
- eager matching: the regex engine stops proceeding further and returns the match
  - white|whitewash -> match(**white**wash)
- GREP: Generalized Regular Expression Pattern Matcher(Ken Thompson)
- NFA: Nondeterministic Finite Automata

- Java String class implements GREP

Lecture9

- Data class encapsulates points in time
- Calendar assigns a name to a point in time

The importance of encapsulation

- even a simple class can benefit from different implementations
- users are unaware of implementation
- **public instance variables would have blocked improvement**
- don't use public fields, even for simple class

Accessor and Mutator

- Mutator: Change object state
- Accessor: Reads object state without changing it
- Class without mutators is immutable
  - String, Day is immutable
  - Date and GregorianCalendar are mutable
- References to **immutable** objects can be **freely shared**
- Don't share **mutable** references
- Good idea to mark immutable instance fields as final
- **final object reference can still refer to mutating object**

Law of Demeter

A method should only use objects that are

- instance fields of its class
- parameters
- objects that it constructs with new

shouldn't use an object that is returned from a method call

Quality of Class Interface

- Cohesion
  - class describes a single abstraction
  - methods should be related to the single abstraction
- Completeness
  - support operations that are well-defined on abstraction
- Convenience
  - a good interface design
- Clarity

    - easy to understand
- Consistency
    - related feathers of a class should have matching

Class Invariants

Condition that is

- true after every constructor
- preserved by every method

Useful for checking validity of operations

- Unit test : test of a single class

Lecture10

Pattern Concept

each pattern has

- a short name
- a brief description of the context
- a lengthy description of the problem
- a prescription of the solution

Design Pattern	Implementation
Iterator Pattern	LinkedList (Java)
MVC Pattern	HTML editor
Observer Pattern	Button (Java)
Strategy Pattern	Collections-Comparator
Adapter Pattern	InputStream -> InputStreamReader
Factory Method Pattern	Create different kinds of iterator

Lecture11

Swing Component

Base of hierarchy: `Component`

Most important subclass: `Container`

Lecture12

- Process: An instance of program running on computer
- Thread: Dispatchable unit of work within process
- Thread Switch: Switching processor between threads within same process

Lecture13

Type and Value

- Type: set of values and the operations that can be applied to the values
- Java Types:
  - Primitive types
  - Class types
  - Interface types
  - Array types
  - `null` types
  - (void is not a type)
- Java Values:
  - value of primitive type
  - reference to object of class type
  - reference to array
  - `null`
  - (Can't have value of interface type)

Subtype Relationship

S is a subtype of T if

- S and T are the same type
- S and T are both class types, and T is a direct or indirect superclass of S
- S is a class type, T is an interface type, and S or one of its superclasses implements T
- S and T are both interface types, and T is a direct or indirect superinterface of S
- S and T are both array types, and the component type of S is a subtype of the component type of T
- S is not a primitive type and T is the type `Object`
- S is an array type and T is `Cloneable` or `Serializable`
- S is the `null` type and T is not a primitive type

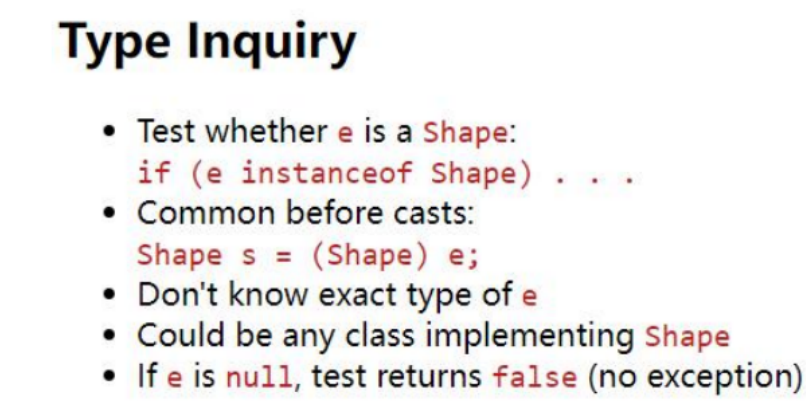
- enum class = public static final xxx
  - it is a class, it can add methods, fields and constructors

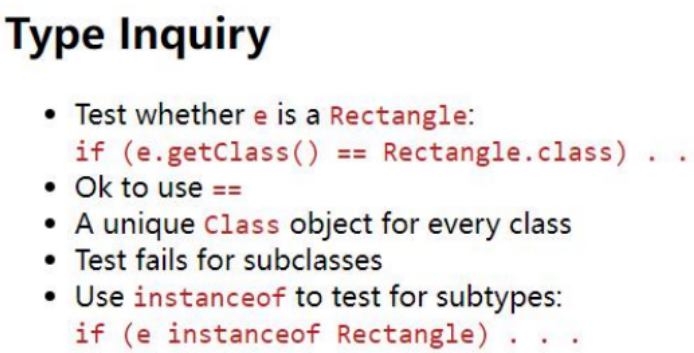
Type Inquiry

```
// Test2 extends Test
Test a = new Test();
Test b = new Test2();
Test2 c = new Test2();
Test d = b;

// new 出来的是什么，它以及它的父类和接口都可以
if(a instanceof Test2) false
if(b instanceof Test) true
if(b instanceof Test2) true
if(c instanceof Test) true
if(d instanceof Test2) true

// new 出来的是什么，就判断是哪个
b.getClass() == Test2.class true
b.getClass() == Test.class false
```





equals

```java
public boolean equals(Object o){
    if (o == this) return true;
    if (o == null) return false;
    // 下面两个选择一个适合的
    if(!(o instanceof XXX class)) return false;
    if(this.getClass != o.getClass) return false;
    // 其它属性的判断
    XXX obj = (XXX) o;
    if(this.A.equals(o.A) && this.b.equals(o.B)) ... return true;
}
```

- if x.equals(y), then x.hashCode() == y.hashCode()
- Object.hashCode hashes memory address

Generics

- cannot replace type variables with primitive type
- cannot construct new objects of genetic type

- Cannot form arrays of parameterized types
  `Comparable<E>[]` is illegal. Remedy: `ArrayList<Comparable<E>>`
- Cannot reference type parameters in a static context (static fields, methods, inner classes)
- Cannot throw or catch generic types
- Cannot have type clashes after erasure. Ex. `GregorianCalendar` cannot implement `Comparable<GregorianCalendar>` since it already implements `Comparable<Calendar>`, and both erase to `Comparable`

Java Bean

- Java Component Model
- Bean has
  - method
  - properties
  - events

- Property = value that you can get and/or set
- Most properties are get-and-set
- Can also have get-only and set-only
- Property *not the same as* instance field
- Setter can set fields, then call repaint
- Getter can query database

Exception

assertions throws `AssertionError` is condition false and checking enables: unchecked exception

the socket constructor throws an `UnknownHostException` if it can't find the host: checked exception

- use try-with resource

- `Rectangle[]` is a subtype of `Shape[]`
- Can assign `Rectangle[]` value to `Shape[]` variable:

```
Rectangle[] r = new Rectangle[10];
Shape[] s = r;
```

- Both `r` and `s` are references to the same array
- That array holds rectangles
- The assignment
  `s[0] = new Polygon();`
  compiles
- Throws an `ArrayStoreException` at runtime
- Each array remembers its component type

- `ArrayStoreException` : unchecked exception

Object.clone throws `CloneNotSupportException` : checked exception