1. Enum

Link: <https://www.csee.umbc.edu/courses/undergraduate/202/spring12/lectures/enums.pdf>

+ Java Platform SE 7

- Definition: Enumerated values are used to represent a set of named values (enum is object)

- Benefit of enum:

+ Acceptable values are obvious -> must choose one of the enumerated values defined already

+ Type safety -> compiler check type of enum

+ Name-spacing -> every value is name-spaced off of the enum type itself

+ Printable

+ Storage of additional information

+ Retrieve of all enumerated values as an array -> Suit[] suits = Suit.values();

+ Comparison of Enumerated values -> if(suit == Suit.CLUBs)

1. Lambda

+ Java Platform SE 8

- Lambda expression is object

# Lambda1:

- Advantage of Lambda:

+ Concise syntax (ngan ngon)

+ Deficiencies with anonymous inner classes (inner class:bulky, hard to optimize)

+ Convenient for new streams library, support streams

+ Programmers are used to the approach

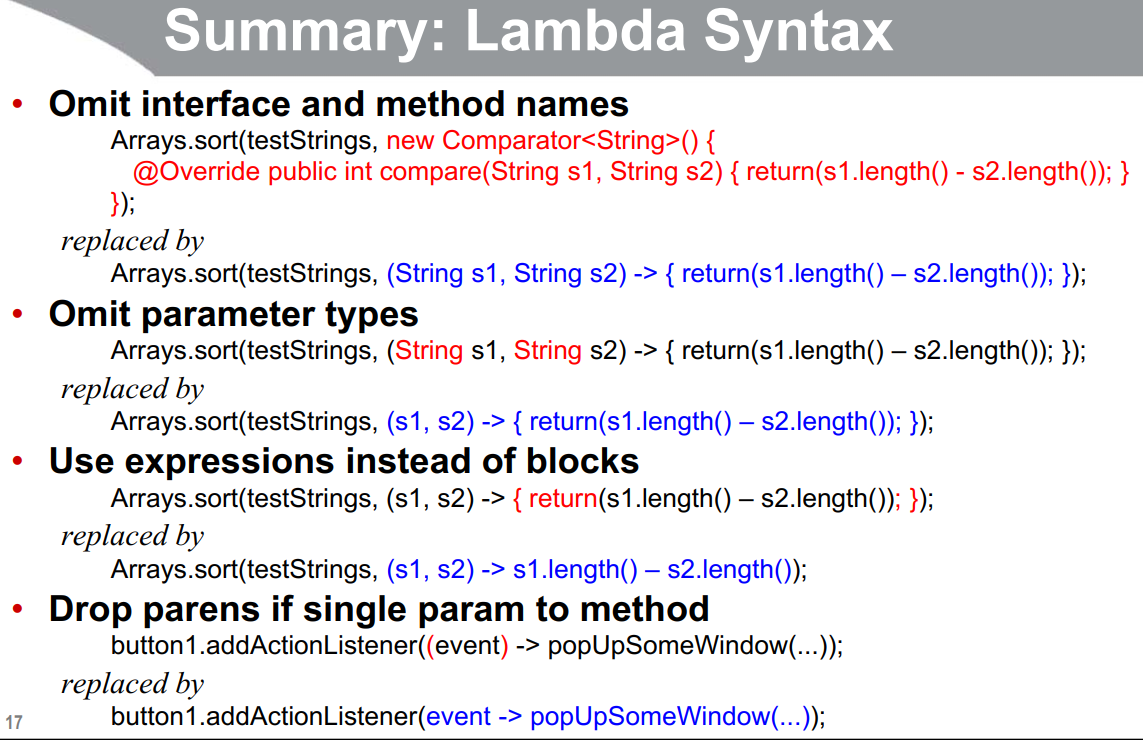
+ Encourage functional programming

- DisAdvantage of lambda:

+ Type of a lambda is class that implements interface, not a “real” function

• Must create or find interface first, must know method name

+ Cannot use mutable local variables



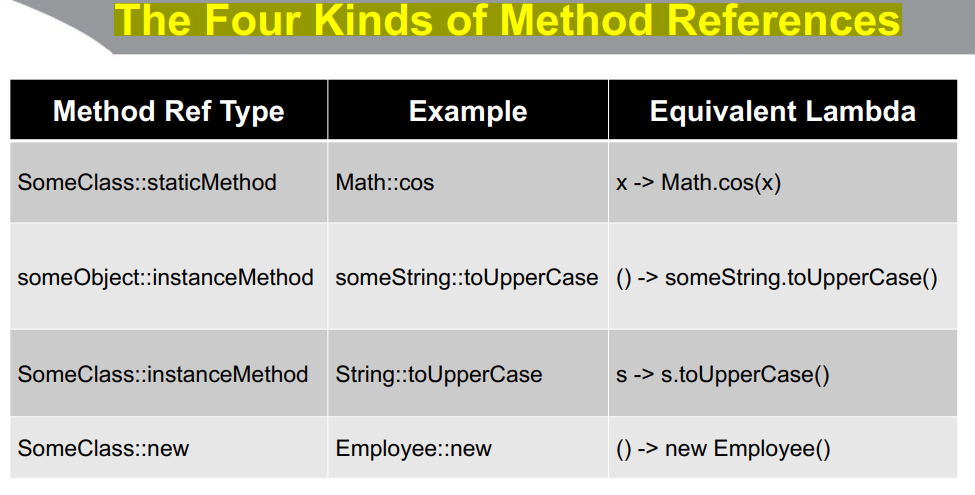
# Lambda2:

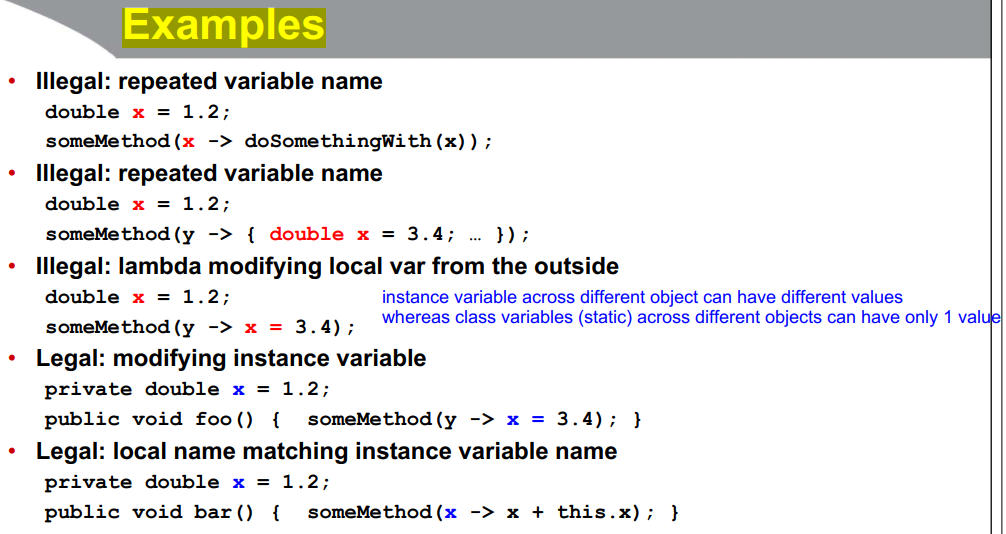
**@FunctionalInterface**

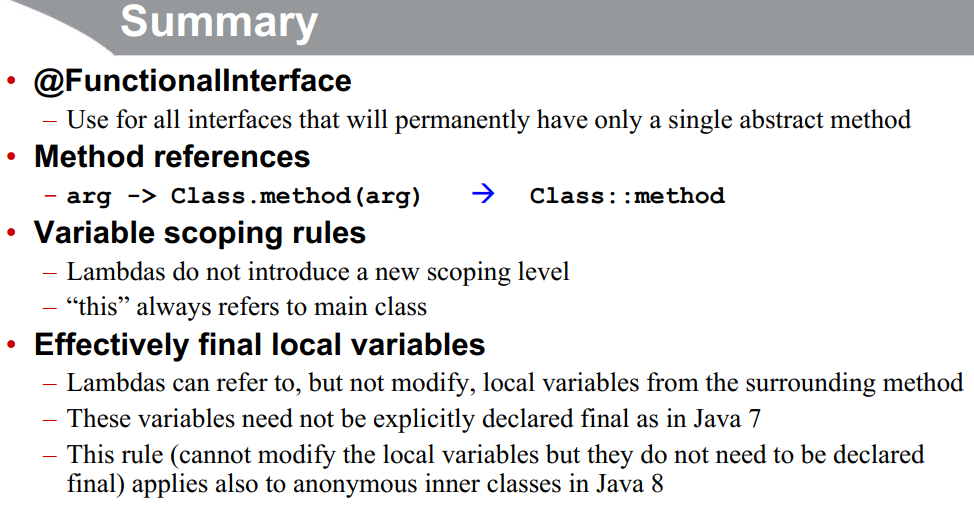
+ Functional Interfaces in Java 8 allows exactly one abstract method inside them.

+ @FunctionalInterface annotation is useful for compilation time checking of your code.

+ This feature in Java, which helps to achieve functional programming approach.







# Lambda3:

* Runnable: Ø → Ø
* Consumer: T → Ø (Lets you make a “function” that takes in a T and no return value)
* IntConsumer: int → Ø
* Supplier: Ø → T (Lets you make a no-arg “function” that returns a T)
* Function: *T*1 → *T*2 (Lets you make a “functions” that takes in a T1 and returns a T2)
* IntFunction: int → T
* IntUnaryOperator: int → int
* BiFunction: (*T*1, *T*2) → *T*3  (Lets you make a “functions” that takes 2 arguments T1,T2 and return T3)
* IntBinaryOperator: (int, int) → int
* Predicate: T → Boolean (lets you make a “function” to test a condition)
* BinaryOperator: (T1,T2) -> T3 ~ BiFunction<T,U,R> where T, U, R are all the same type

1. Stream

- Java 8

- Making streams more powerful, faster, and more memory efficient than Lists

- The three coolest properties:

• Lazy evaluation • Automatic parallelization • Infinite (unbounded) streams

#Stream1:

- 3 ways to make a Stream:

+ From Lists: List<String> words =…; words.stream().map()…

+ From object arrays: Employee[] workers =…; Stream.of(workers).map()…

+From individual elements: Employee[] e1 =…; Employee[] e2 =…; Employee[] e3 =…;

Stream.of(e1,e2,….).map()…

- findFirst(): return Optional<T> -> ~ check the optional is empty?

- Turning Streams into Pre-Java-8 Data Structures -> do this only at the end, after you have done all the stream.

+ Output as a list:

List<String> w = someStream**.collect(Collectors.toList())**

List<Employee> w = someStream**.collect(Collectors.toList())**:

+ Output as an array:

String[] w = someStream.**toArray(String[]::new)**

Employee[] w = someStream.**toArray(Employee[]::new)**

#Stream2:

- **limit**(n) returns a Stream of the first n elements.

- **skip**(n) returns a Stream starting with element n (i.e., it throws away the first n elements)

- reduce(starterValue, binaryOperator): ~ reduce(baseValue,Integer:sum())

1. Serialization
2. Network

Link: <https://www.slideshare.net/tusharkute/network-programming-in-java>

Link: <https://slideplayer.com/slide/5150902/>

Client <-> Network <-> Server

**Socket**

- Sockets provide an interface for programming networks at the transport layer-> Network communication using Socket as I/O.

- Socket is endpoint for communication between two machines.

- Socket-based communication can communicate on program in Java or Non-Java.

- Socket uses TCP to communicate over the network

**Constructor:**

Socket(String remoteHost, int remotePort)

Socket(InetAddress ip, int remotePort)

**TCP-Transmission Control Protocol**

- TCP provides a reliable flow of data between 2 computers (point-to-point).

Ex: HTTP,FTP, Telnet require a reliable communication channel.

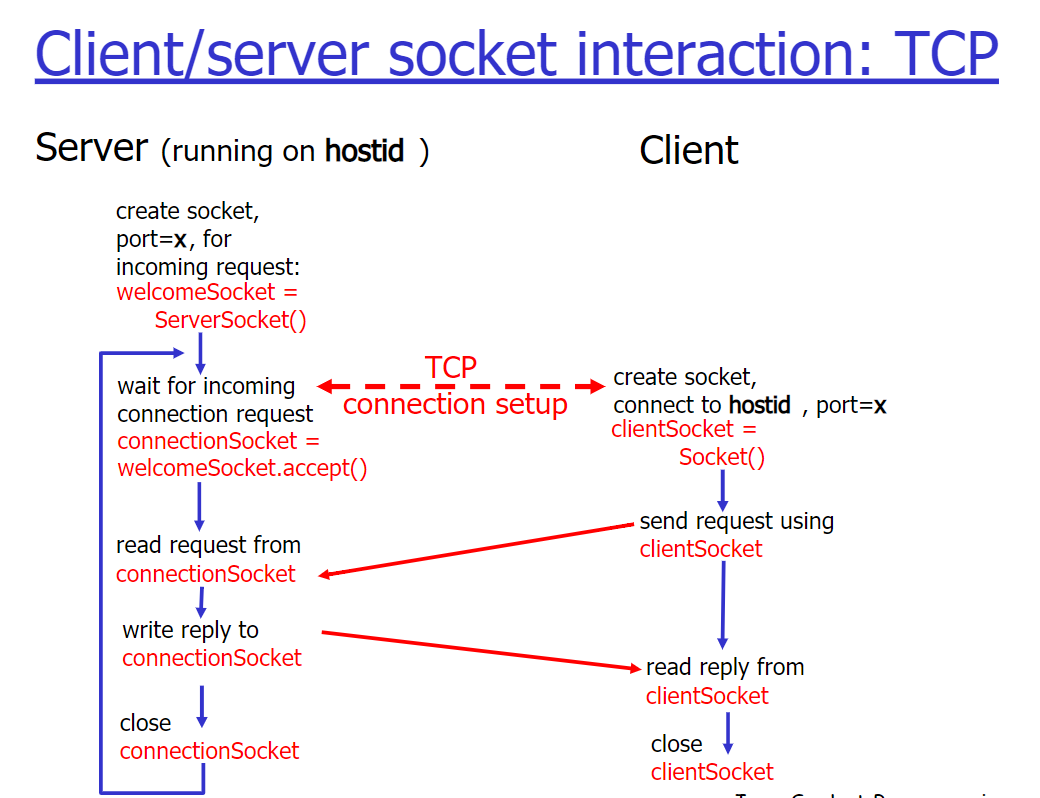
- The *URL,URLConnection, Socket, ServerSocket* classes all use Transmission Control Protocol (TCP) to communicate over the network

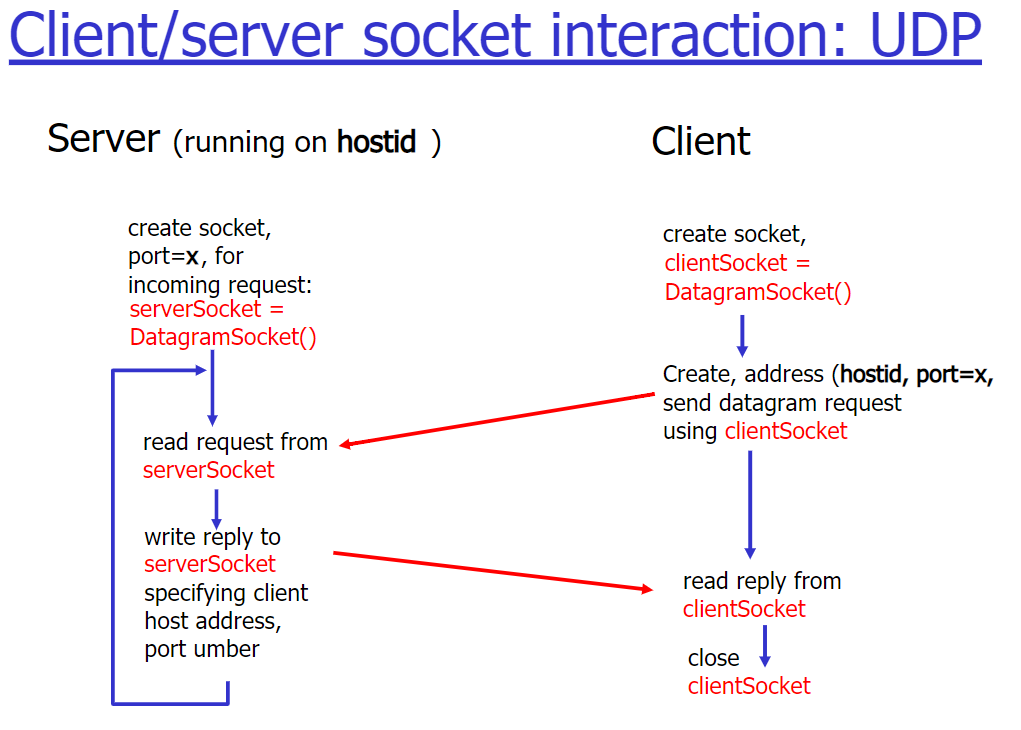
**UDP-User Datagram Protocol**

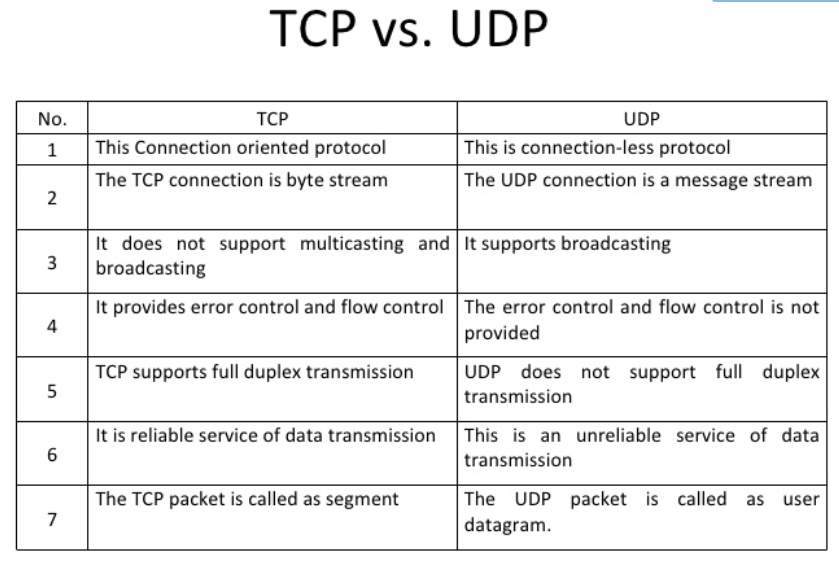
- UDP sends independent packets of data (called datagrams) from one computer to another with no guarantees about arrival -> not reliable, but good in speed and cost

Ex: streaming media, games, Internet telephony…

- The *DatagramPacket, DatagramSocket, MulticastSocket* classes are for use with User Datagram Protocol (UDP)







Port # IP

**Port**:

- TCP and UDP use Ports to deliver the data to the right application

- 16 bit integer value (2^16), 0 – 1023 (well-known ports) to 65535

- FTP (20,21); TELNET (23); SMTP (25); POP3 (110); HTTP (80); DSN(53)

**IP**:

- IP as address 32bit

- IP uses to deliver data to the right computer on the network

- Java.net.InetAddress-> both IP address and domain name

1. Thread