上过的课程在这里：
存档

Class 24: C++ vs. Java, Garbage collection

Difference Between Java & C++

1) **Platform compatible, write once, compile once, run everywhere on JVM**
    a) C++: write once, **compile** everywhere
    b) What is JVM? Java Virtual Machine.
    c) JRE vs. JDK
    - JRE (Java Runtime Environment) is the JVM program, Java application need to run on JRE
    - JDK contains the tools for developing Java programs running on JRE, for example, it provides the compiler "javac"
2) Compiles to Java byte code can be recognized by JVM, independent to underline OS.
3) Interpreter to routines on systems with different machine codes.
    a) Compile/Interpret Language?
4) Strongly encouraged **Object-Oriented Programming Paradigm**, everything in Java is class/Object.
5) All types(reference types, primitive types) are always **pass by value**.
6) Java does not support unsigned integers.
7) Pointers vs. References.

a) no pointer arithmetics

8) No operator overloading. "+", "-".....

9) Classes/Objects are always allocated on the Heap, there is no way to allocate objects on Stack.

a) "new"

10) Garbage Collection.

b) So we do not need to care about memory management, really?

c) Memory leak?

11) Single inheritance, multiple inheritance only be done by implementing multiple interfaces.

1), 2), 3)

JVM - The Java Virtual Machine (JVM) is an abstract computing machine. The JVM is a program that looks like a machine to the programs written to execute in it

JRE vs. JDK

- JRE is the JVM program, Java application need to run on JRE
- JDK contains the tools for developing Java programs running on JRE, for example, it provides the compiler "javac"

C++: The source code need to be compiled to directly the machine instructions on which machine the program is running on. The compiled byte code is different on different machines.

8) No operator overloading. "+", "-".....

9) Classes/Objects are always allocated on the Heap, there is no way to allocate objects on Stack.

a) "new"

10) Garbage Collection.

b) So we do not need to care about memory management, really?

c) Memory leak?

11) Single inheritance, multiple inheritance only be done by implementing multiple interfaces.

1), 2), 3)

JVM - The Java Virtual Machine (JVM) is an abstract computing machine. The JVM is a program that looks like a machine to the programs written to execute in it

JRE vs. JDK

- JRE is the JVM program, Java application need to run on JRE
- JDK contains the tools for developing Java programs running on JRE, for example, it provides the compiler "javac"

C++: The source code need to be compiled to directly the machine instructions on which machine the program is running on. The compiled byte code is different on different machines. (machine need to recognize the compiled code)

Java: The source code just need to compiled once and it can run on any machine with JRE installed. (JRE, not the machine need to recognize the compiled code)

Compile/Interpret language:

1). Java source code -> JVM readable byte code (java → class)

It is nearly the same for C++ and Java for "new" to allocate space and create objects on Heap, the difference is about how to manage the dynamic created objects.

C++:

- "new" to create object on heap
- "delete" to reclaim the allocated memory, and this is needed to be done by developers themselves! (imagine how hard it is...)

Java:

- Garbage Collection is the mechanism to automatically detect/delete the unused objects, so that developers do not need to care too much about each individual object's life cycle.

4 Key Points

How do we solve the problem? How does the GC work in Java?

- **What is GC component**
- **Steps to finish the job**
- **generational mechanism, why?**
- **how to determine what objects can be collected**

Tutorial about Java GC.
http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html

Concurrency vs. Parallel
Process vs. Thread
Synchronization and race
Mutual exclusion, critical section, and lock
Condition synchronization
Atomicity

Concurrency vs. Parallel

Concurrency: multiple tasks run *simultaneously*. Semantic concept, the "time" here may be an abstract concept. You can't tell who's first, who's second. You can't tell the order in general.
Example: Friends eating together, two people leaving office at the same time, etc…
In a Java program: maybe a before b, maybe b before a, maybe same time---I don't know the order.

Parallel: multiple tasks *physically* run simultaneously. Implementation level concept. In real time, there are at least two executors.
Example: multilane highway, multicore machines, Hadoop clusters, …

Now think: do we need to take care of concurrency on a single core machine?
Yes! Concurrency is in the program, parallelism is in the actual execution.
If in the program, event A is concurrent to event B, A single core machine may:

1. Execute half of A
2. Execute B
3. Execute the second half of A

The ways to perform parallel programming (to physically launch multiple executors)

Parallel: multiple tasks *physically* run simultaneously. Implementation level concept. In real time, there are at least two executors.
Example: multilane highway, multicore machines, Hadoop clusters, …

Now think: do we need to take care of concurrency on a single core machine?
Yes! Concurrency is in the program, parallelism is in the actual execution.
If in the program, event A is concurrent to event B, A single core machine may:

1. Execute half of A
2. Execute B
3. Execute the second half of A

The ways to perform parallel programming (to physically launch multiple executors)

Multiprocess vs. multi-thread

Process: an independent execution of instructions with independent memory space, stack, heap, and os resource.
Each process sees a complete memory space (pretend to be the only task of a system.)
Different processes communicates through interprocess communication (explicit IPC).

there are at least two executors.
Example: multilane highway, multicore machines, Hadoop clusters, …

Now think: do we need to take care of concurrency on a single core machine?
Yes! Concurrency is in the program, parallelism is in the actual execution.
If in the program, event A is concurrent to event B, A single core machine may:

1. Execute half of A
2. Execute B
3. Execute the second half of A

The ways to perform parallel programming (to physically launch multiple executors)

Multiprocess vs. multi-thread

Process: an independent execution of instructions with independent memory space, stack, heap, and os resource.
Each process sees a complete memory space (pretend to be the only task of a system.)
Different processes communicates through interprocess communication (explicit IPC).

Thread: an independent execution of instructions with **shared memory space**.
Each thread has its private: stack, program counter and register states.
Thread in the same process has shared: heap, static memory segment, os resource
Communication performed through shared memory read/writes. ← here's why we need to discuss concurrency!
Lightweight data and resource sharing (compare to process).

Semantically, the fundamental difference between a "process" and a "thread" is *if they have independent memory space*. Implementation-wise, there are corner cases. Be careful about your wording here.

Different processes communicates through interprocess communication (explicit IPC).

Thread: an independent execution of instructions with **shared memory space**.
Each thread has its private: stack, program counter and register states.
Thread in the same process has shared: heap, static memory segment, os resource
Communication performed through shared memory read/writes. ← here's why we need to discuss concurrency!
Lightweight data and resource sharing (compare to process).

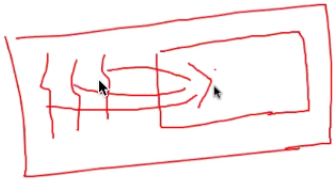
Semantically, the fundamental difference between a "process" and a "thread" is *if they have independent memory space*. Implementation-wise, there are corner cases. Be careful about your wording here.

Examples: multiple roads vs. multiple lanes.
Consider: Communication overhead? Resource isolation (fault tolerance)? Creation/destroy overhead?

Java concurrency is focusing on multithread cases in real life. A Java application runs by default in one process. Within a Java application you work with several threads to achieve parallel processing or asynchronous behavior.

Not properly designed parallel programs:

https://www.youtube.com/watch?v=MNhubpzhs-Q

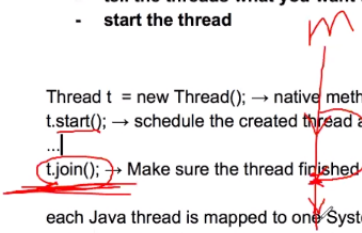


Java Thread

What is Thread in Java? How can we create threads and control their behaviors?

In Java, everything is an OBJECT, so is Thread! If you need to create new threads helping you make your program concurrently, you have to

- **create the threads objects**
- **tell the threads what you want them to do**
- **start the thread**



Thread t = new Thread(); → native method call, created one System Thread
t.start(); → schedule the created thread and make ready to go
...
t.join(); → Make sure the thread finished after this line.

each Java thread is mapped to one System thread
(what is system thread? -> managed and scheduled by OS kernel rather than in user space)

```
class Thread {
  public void run() {
    // what this thread need to do.
  }
  public void start() { // when this method is called, the thread is ready for scheduling.
    // … prepare...
    run();
```

If two "conflicting operations" are in different threads and are not properly synchronized (concurrent), they will introduce ***data races***. In general, two operations conflict with each other if they operate on the same memory location, and at least one of them is a write. Races are mostly treated as bugs in Java programs.

So to form a data race there should be 3 factors:

1. More than one operations work on the same memory location
2. At least one operation is a write
3. At least two of those operations are concurrent

Example 1
int a = 10;
Thread 1:
    a = 8;
Thread 2:
    int b = a + 1;
Is there a race?
(yes!)

Example 2
int a = 10;
Thread 1: