1. Runtime polymorphism
2. Compile time polymorphism
3. Overloading is beneficial than overriding:
4. JVM internal architecture:
   1. JVM is machine where your byte code is taken as input and then it executes the byte code.
   2. Major benefit is WORE(Write once Run Everywhere)
      1. Platform independent because the code is converted into byte
   3. Divided into
      1. Class loader subsystem:
         1. Loading part: Loads the class from different jars to particular program
            1. Bootstrap class loader:

Loads classes while loading jvm

* + - * 1. Extension class loader:

JRE.lib when you install Java

RT.jar will consist all classes used by java

* + - * 1. Application class loader:

Used for loading class path for particular application

* 1. Memory in JVM
     1. Method area
     2. Heap area
     3. Stack area
        1. Multithreaded environment
     4. PC registers:
        1. Each thread has separate pc register
        2. Holds address for current thread
     5. Execution engine
        1. Interpreter
           1. Interprets byte code
           2. Executes code line by line
        2. JIT compiler
           1. Compiles the entire byte code
        3. Garbage collector:
           1. Cleans memory
           2. Removes unused
        4. Native method library:
           1. Library that java uses
        5. JNI:
           1. Provides interaction between this library and native

1. Two places to store string:
   1. String pool
      1. String s = “Java” – Literal
      2. String s2 = “Java”
      3. Since both are same s2 will only point to s.
   2. Heap
      1. String s1 = new String(original: “java”);
      2. String s3 = new String(original: “java”);
      3. Creates two different objects are stores in heap
   3. All strings are immutable if you create using string literal.
      1. In the above example of string pool if we use s = “java1” new object will java1 is created and s is pointed to that object. “Java” will still be in string pool.
   4. String objects are stored in heap memory so all objects are different from one another.
   5. String buffer and string builder are mutable in nature.
   6. String buffer:
      1. Used when you want thread safe
      2. Slower because synchronized
   7. String builder:
      1. Used in normal application
      2. Faster because not synchronized
      3. StringBuilder stringBuilder = new StringBuilder();
      4. stringBuilder.append(“Java”).append(“oops”);
      5. System.out.println(stringBuilder);
      6. Output: Javaoops
2. Difference between interface and abstract classes
   1. Both are used to hide implementation to outside world
   2. Interface:
      1. In a class if all methods are abstract method i.e. only method header.
      2. All methods inside interface is public
   3. Abstract classes:
      1. Can have both defined method and abstract method.