类、抽象类与接口 实例化的类(构造函数)

继承 实现

多态(重载 重写)

泛型

异常捕获

GC

反射

Data in Java

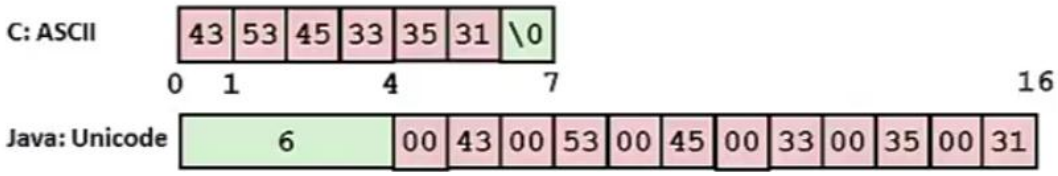
- **Integers, floats, doubles, pointers – same as C**
  - Yes, Java has pointers – they are called 'references' – however, Java references are much more constrained than C's general pointers
- **Null is typically represented as 0**
- **Characters and strings**
- **Arrays**
- **Objects**

1. 整型，浮点型 – 和C是一样的
2. java有指针，但是和C里的说法不一样，java把指针称之为引用，和C不一样的是，java的引用
   总是指向一个对象的开始，也就是一个数据结构体的开头
3. null == 0

Data in Java

- **Characters and strings**
  - Two-byte Unicode instead of ASCII
    - Represents most of the world's alphabets
  - String not bounded by a '\0' (null character)
    - Bounded by hidden length field at beginning of string

the string 'CSE351':



4. 字符与字符串

java使用的字符是unicode编码而不是ASCII编码

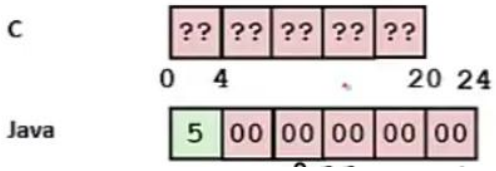
C字符串以\0开头

java的头四个字节表示字符串的长度

Data in Java

- **Arrays**
  - Every element initialized to 0
  - Bounds specified in hidden fields at start of array (int – 4 bytes)
    - *array.length* returns value of this field
    - *Hmm, since it has this info, what can it do?*

int array[5]:



arrays.

5. 数组

java 每个元素都被默认初始化为0, 和字符串一样,java开始的

四个字节标识数组的长度

作用：检查溢出(out of bounds)

Data structures (objects) in Java

- **Objects (structs) can only include primitive data types**
  - Include complex data types (arrays, other objects, etc.) using *references*

C

```
struct rec {
  int i;
  int a[3];
  struct rec *p;
};
```

Java

```
class Rec {
  int i;
  int[] a = new int[3];
  Rec p;
…
};
```

java： 对象只能包含简单数据类型，不能包含复杂数据类型(对于java就是对象中不能包含对象，而只能是对象的引用)

Data structures (objects) in Java

- **Objects (structs) can only include primitive data types**
  - Include complex data types (arrays, other objects, etc.) using *references*

C

```
struct rec {
  int i;
  int a[3];
  struct rec *p;
};
```

Java

```
class Rec {
  int i;
  int[] a = new int[3];
  Rec p;
…
};
```

```
struct rec *r = malloc(...);
struct rec r2;
r->i = val;
r->a[2] = val;
r->p = &r2;
```

```
r = new Rec;
r2 = new Rec;
r.i = val;
r.a[2] = val;
r.p = r2;
```

Pointers/References

- Pointers in C can point to any memory address
- References in Java can only point to an object
  - And only to its first element – not to the middle of it

C
```c
struct rec {
  int i;
  int a[3];
  struct rec *p;
};
some_fn(&(r.a[1]))  //ptr
```

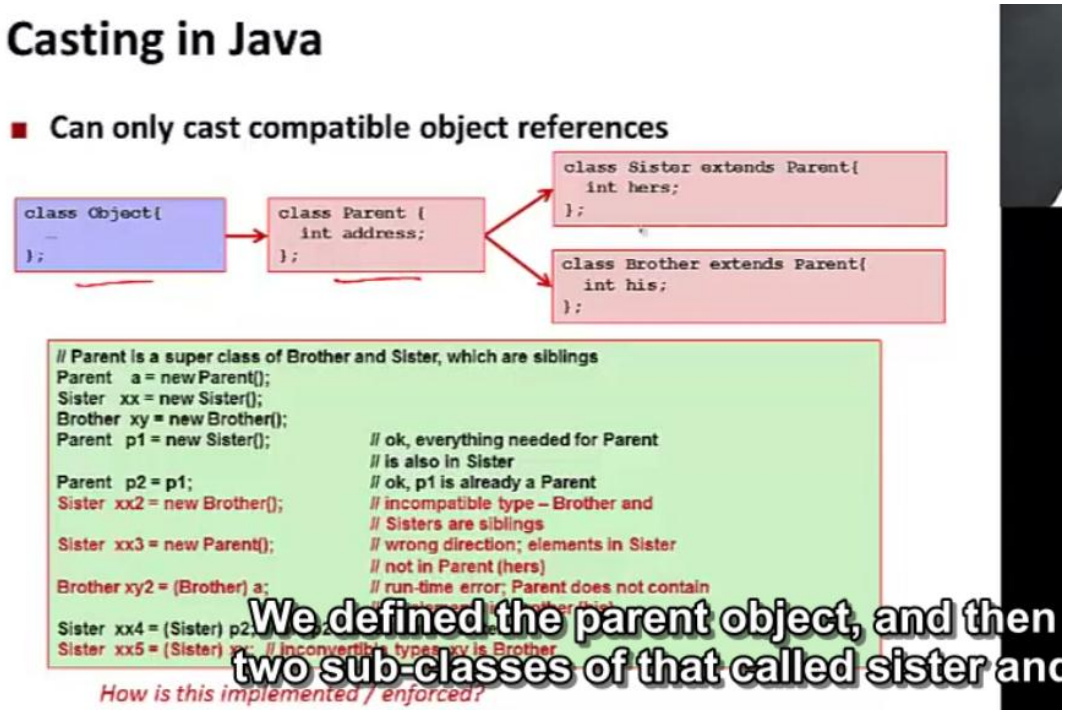
Java
```java
class Rec {
  int i;
  int[] a = new int[3];
  Rec p;
…
};
some_fn(r.a, 1)  //ref & index
```

再次强调：java 里的引用只能指向对象的首元素

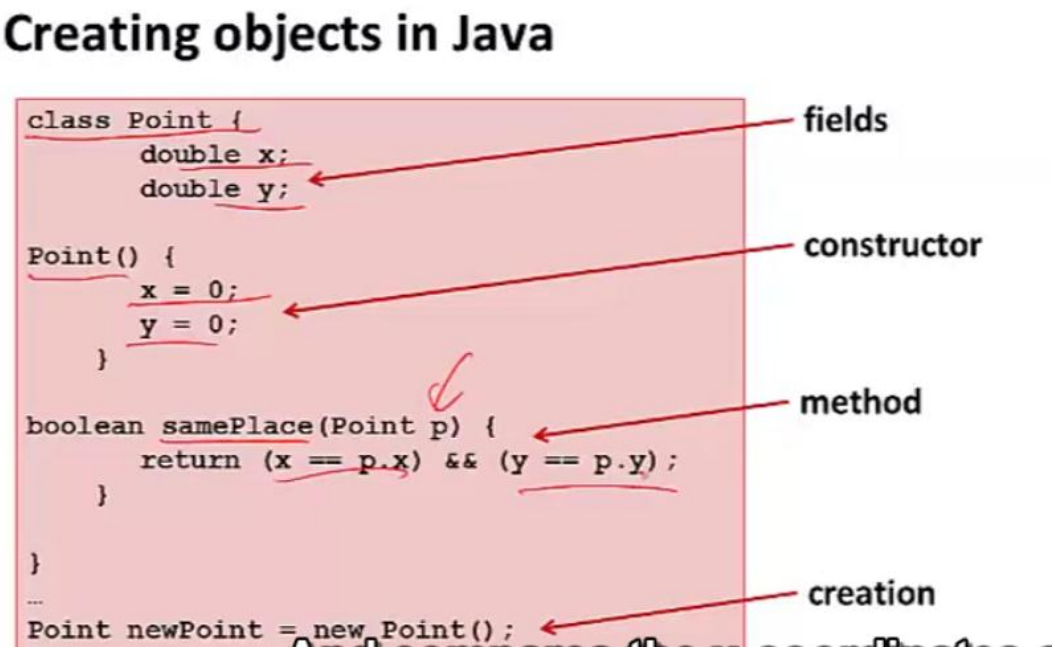
Pointers to fields

- In C, we have "->" and "." for field selection depending on whether we have a pointer to a struct or a struct
  - (*r).a is so common it becomes r->a
- In Java, *all variables are references to objects*
  - We always use r.a notation
  - But really follow reference to r with offset to a, just like C's r->a

java所有对象都是引用，简单的用.来解引用
C里的指针能被强转为任何其他类型的指针



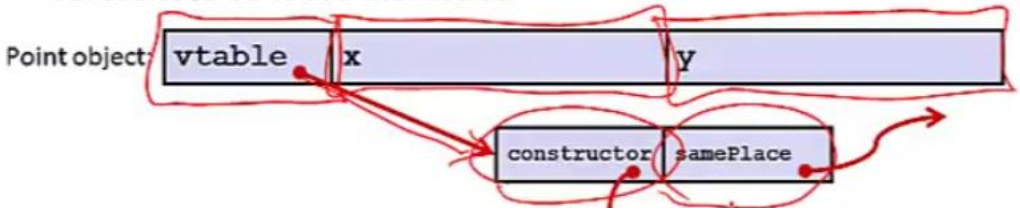
1. 向上转型会丢失一些信息
2. 父类没有子类的一些信息，不能强转，否则会抛异常
3. 引用指向的原始对象是强转类的向下类型时，可强转



1. 内存申请一个类返回一个引用同时执行构造函数

Creating objects in Java

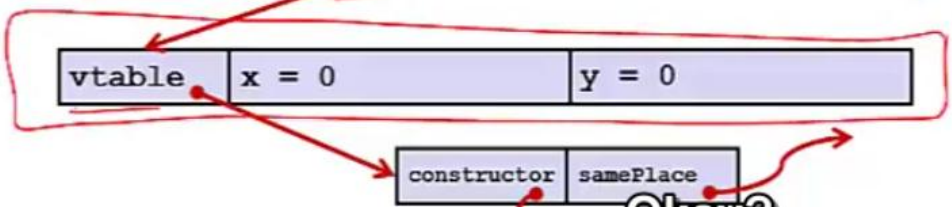
- **"new"**
  - Allocates space for data fields
  - Adds pointer in object to "virtual table" or "vtable" for class
    - vtable is shared across all objects in the class!
    - Includes space for "static fields" and pointers to methods' code
  - Returns reference (pointer) to new object in memory
  - Runs "constructor" method
- **The new object is eventually garbage collected if all references to it are discarded**



二级引用

vtable不会释放

- **Constructor code is found using the 'vtable pointer' and passed a pointer to the newly allocated memory area for newPoint so that the constructor can set its x and y to 0**
  - Point.constructor( )



执行构造函数并将内存首地址传入构造函数

- **Methods in Java are just functions (as in C) but with an extra argument: a reference to the object whose method is being called**
  - E.g., newPoint.samePlace calls the samePlace method with a pointer to newPoint (called 'this') and a pointer to the argument, p – in this case, both of these are pointers to objects of type Point

- Method becomes Point.samePlace(Point this, Point p)
  - return x==p.x && y==p.y; becomes something like:
    return (this->x==p->x) && (this->y==p->y);

父类

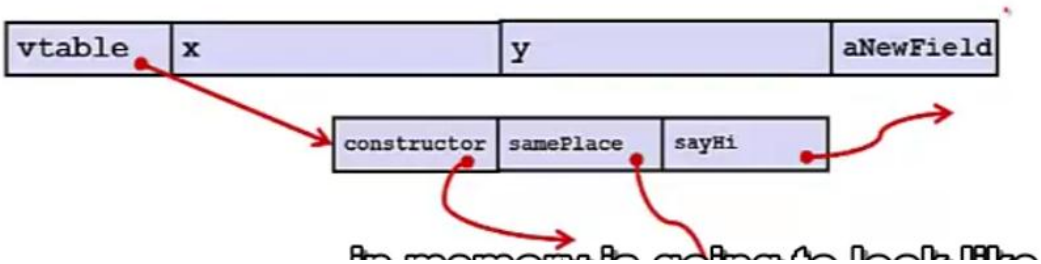
Subclassing

```
class PtSubClass extends Point{
  int aNewField;
  boolean samePlace(Point p2) {
     return false;
  }
  void sayHi() {
    System.out.println("hello");
  }
}
```

- **Where does "aNewField" go?**
  - At end of fields of Point – allows easy casting from subclass to parent class!
- **Where does pointer to code for two new methods go?**
  - To override "samePlace", write over old pointer
  - Add new pointer at end of table for new method "sayHi"

Subclassing

```
class PtSubClass extends Point{
  int aNewField;
  boolean samePlace(Point p2) {
     return false;
  }
  void sayHi() {
    System.out.println("hello");
  }
}
```



和父类的vtable不一样，但是也是重用的

Implementing Programming Languages

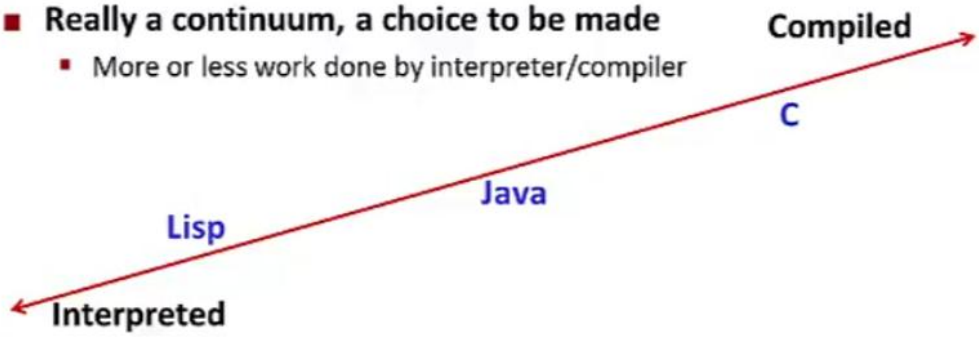
- **Many choices in how to implement programming models**
- **We've talked about compilation, can also *interpret***
  - Execute line by line in original source code
  - Less work for compiler – all work done at run-time
  - Easier to debug – less translation
  - Easier to run on different architectures – runs in a simulated environment that exists only inside the *interpreter* process
- **Interpreting languages has a long history**
  - Lisp – one of the first programming languages, was interpreted
- **Interpreted implementations are very much with us today**
  - Python, Javascript, Ruby, Matlab, PHP, Perl, ...

java是一门解释性的语言

缺点：没有编译器的优化，也就没有debug/release -O2的概念

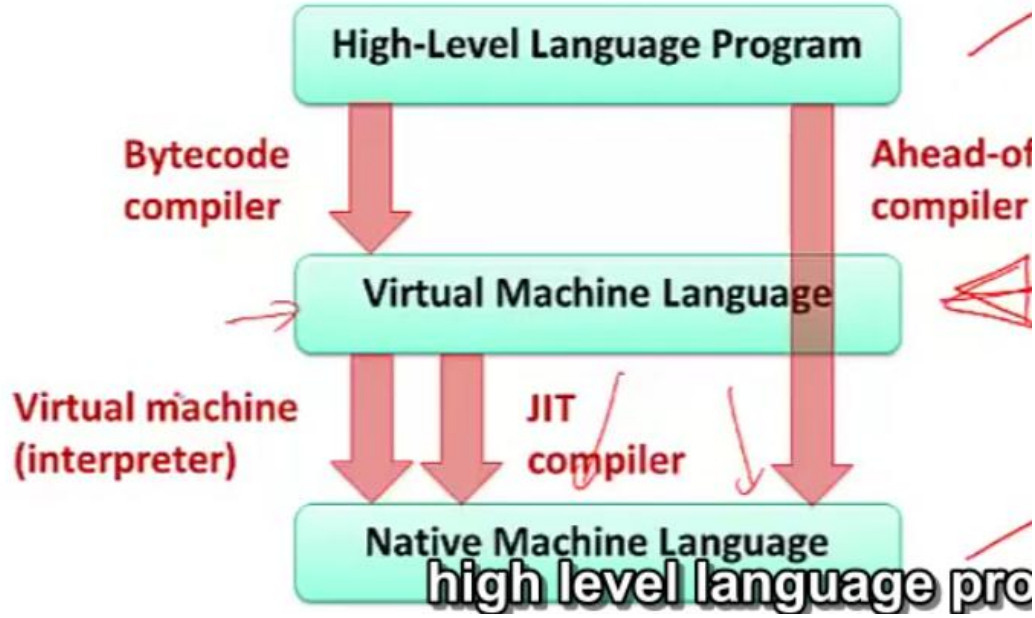
Interpreted vs. Compiled

- **Really a continuum, a choice to be made**
  - More or less work done by interpreter/compiler



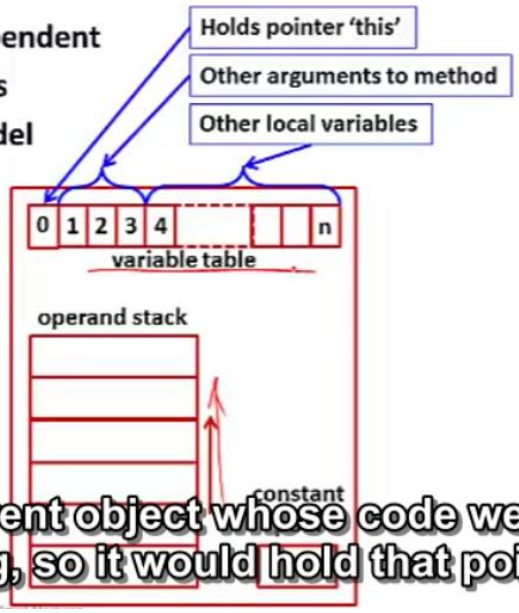
- **Java programs are usually run by a *virtual machine***
  - VMs interpret an intermediate, "partly compiled" language called *bytecode*
- **Java can also b**
  C program is)

Virtual Machine Model

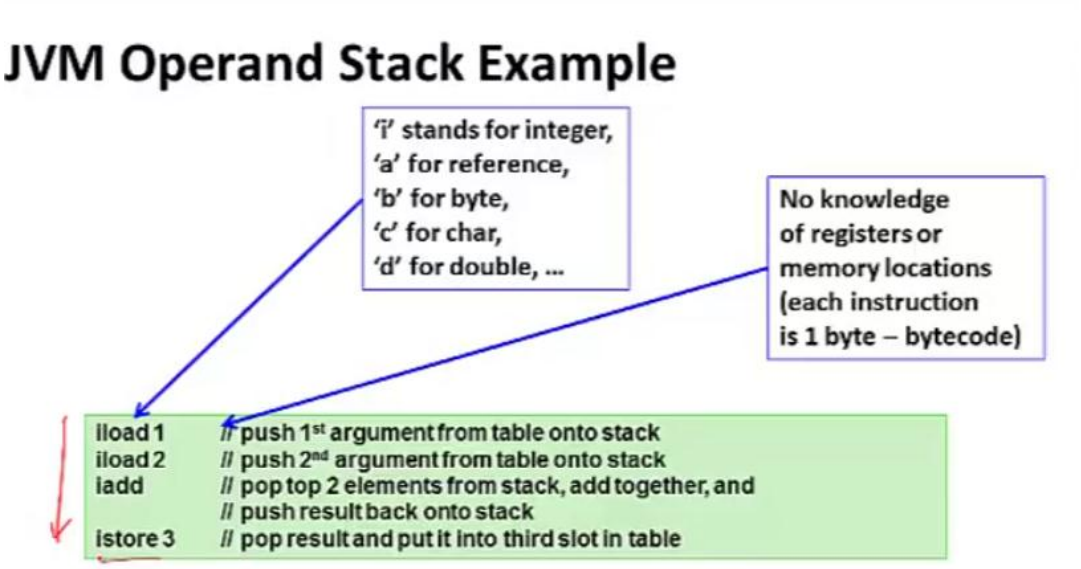


Java Virtual Machine

- Makes Java machine-independent
- Provides strong protections
- Stack-based execution model
- There are many JVMs
  - Some interpret
  - Some compile into assembly
  - Usually implemented in C



the current object whose code we're executing, so it would hold that pointer



没有寄存器的概念

```
public static final void main(String[] args)
{
 int i = 1;
 i = 2;
 System.out.println(i);
 return;
}
```

```
public static final void main(java.lang.String[]);
  Code:
     0: iconst_1
     1: istore_1
     2: iconst_2
     3: istore_1
     4: getstatic     #5                  // Field java/lang/System.out:Ljava/io/PrintStream;
     7: iload_1
     8: invokevirtual #6                  // Method java/io/PrintStream.println:(I)V
    11: return
```