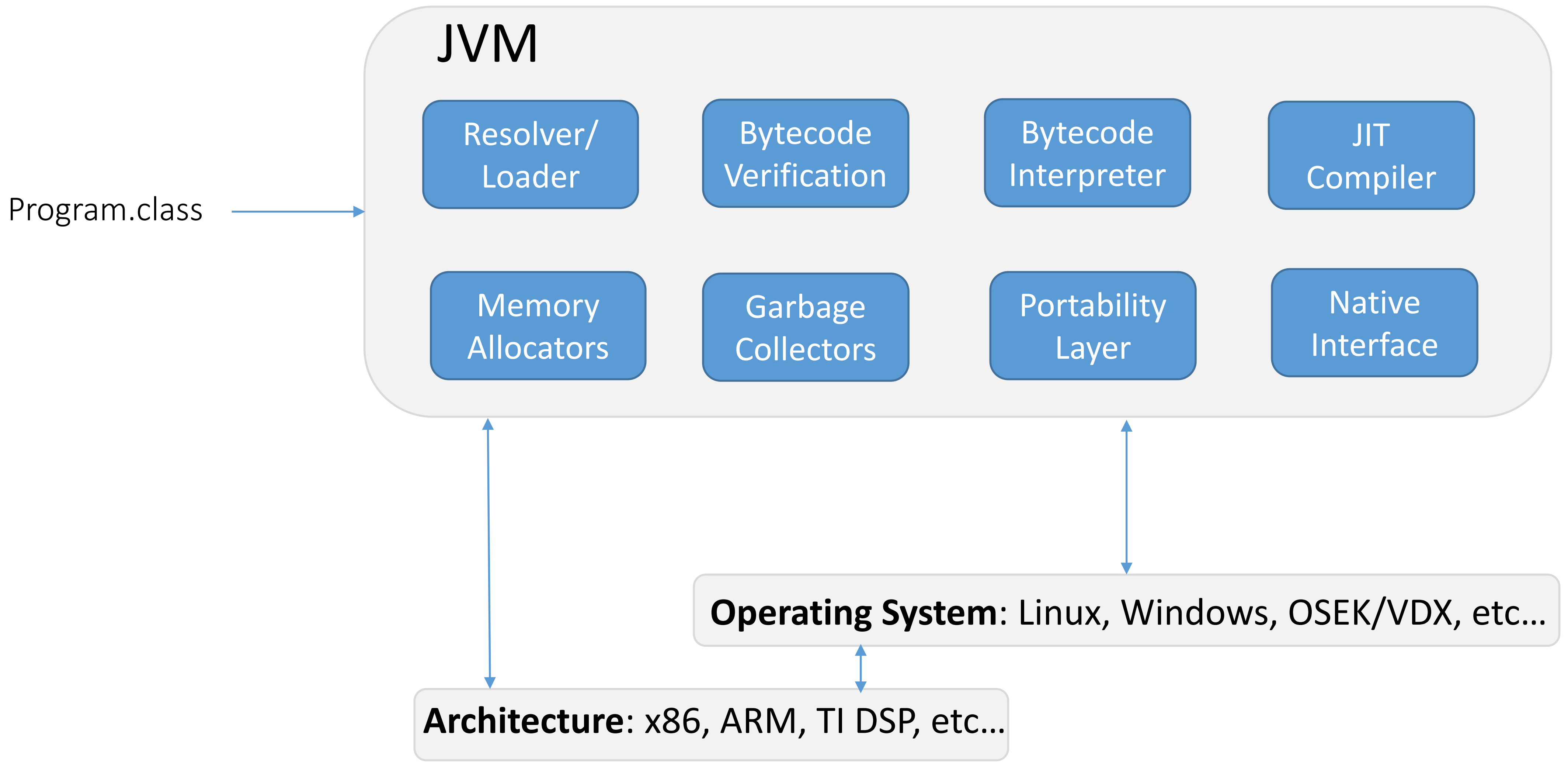
Key JVM Components



```
class Test {
  static int x;
  double d;

  Test();
    Code:
       0: aload_0
       1: invokespecial #1    // Method java/lang/Object."<init>": ()V
       4: return

  public static native int print(double);

   public double pp(int);
    Code:
       0: iload_1
       1: i2d
       2: dreturn
```

Constructor for class Test

Pushes content of local variable 0 (note: the variable is of a reference type) to the stack.

Invoke constructor for the superclass of Test, that is, java.lang.Object…and clear the stack.

Native method. Its implementation could be provided for example in a C/C++ library.

Pushes content of local variable 1 (type integer) to stack.

convert the integer on the stack to a double.

Pop value from stack and return it.

```
static {};
    Code:
       0: sipush      2018
       3: putstatic    #5          // Field x:I
       6: return
}
```

JVM invokes this code **before** main()

push constant 2018 of type short (hence: si) to stack

pop 2018 from stack and write it to static field x.

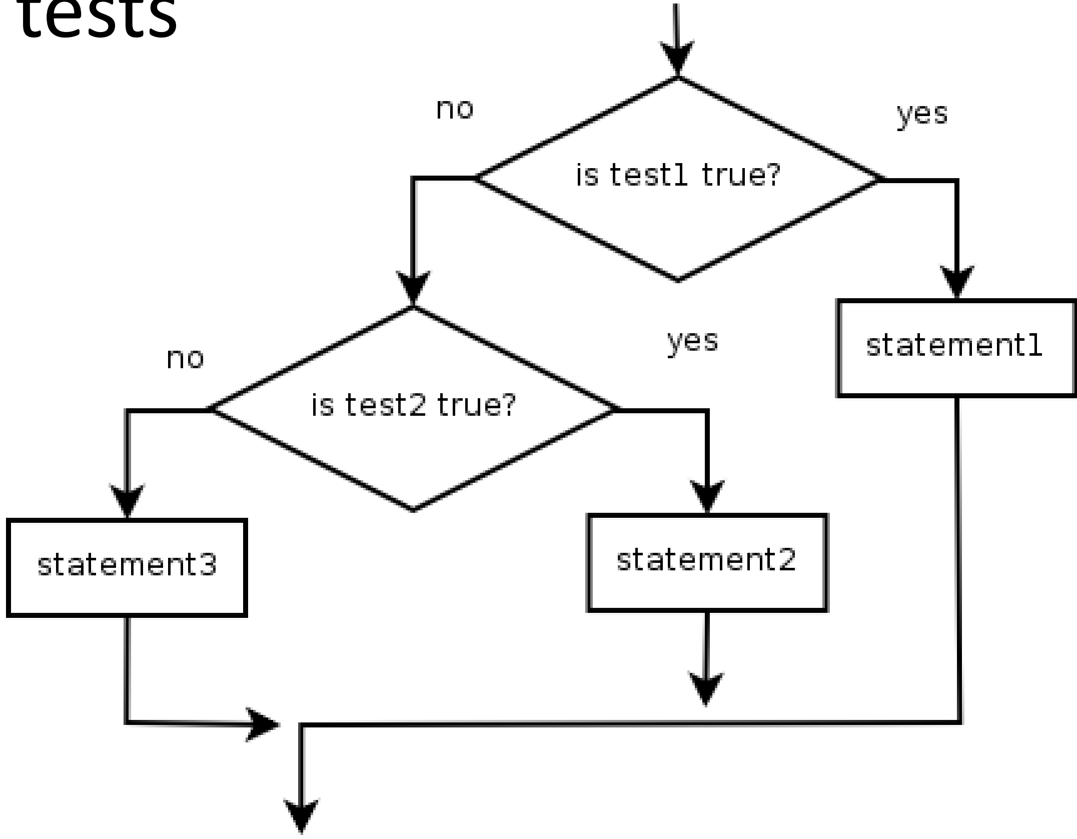
Different kinds of errors

1. Compiler errors
2. Runtime errors
3. Logic errors

Nested `if/else`

Chooses between outcomes using many tests

```
if (test) {
    statement(s);
} else if (test) {
    statement(s);
} else {
    statement(s);
}
```



Example:

```
if (x > 0) {
    System.out.println("Positive");
} else if (x < 0) {
    System.out.println("Negative");
} else {
    System.out.println("Zero");
}
```

Tip: in parallelism/concurrency…try to have the if /else's read from a local variable.

Arrays, Strings, Identity and Equality, (Im)Mutability and Optimizations

```
int[] a1 = new int[] {1,2,3};
int[] a2 = new int[] {1,2,3};

System.out.println("a1 == a2? " + (a1 == a2));
System.out.println("a1.equals(a2)? " + a1.equals(a2));
System.out.println("Arrays.equals(a1, a2)? " + Arrays.equals(a1, a2));

String s1 = "ETH";
String s2 = "ETH";
// String s2 = s1.charAt(0) + "TH";

System.out.println("s1 == s2? " + (s1 == s2));
System.out.println("s1.equals(s2)? " + s1.equals(s2));
```

Language features vs. parallelism: Guidelines

- Keep variables as ‘local’ as possible: global variables means they can be accessed by various parallel activities. While when its local to the process/thread, we are safe against inadvertent accesses to the variable.
- If possible, avoid aliasing of references: aliasing can lead to unexpected updates to memory through a process that accesses a seemingly unrelated variable (named differently).
- If possible, avoid mutable state, in particular when aliased: aliasing is no problem if the shared object is immutable, but concurrent mutations can make bugs *really* hard to reproduce and investigate (“Heisenbugs”)