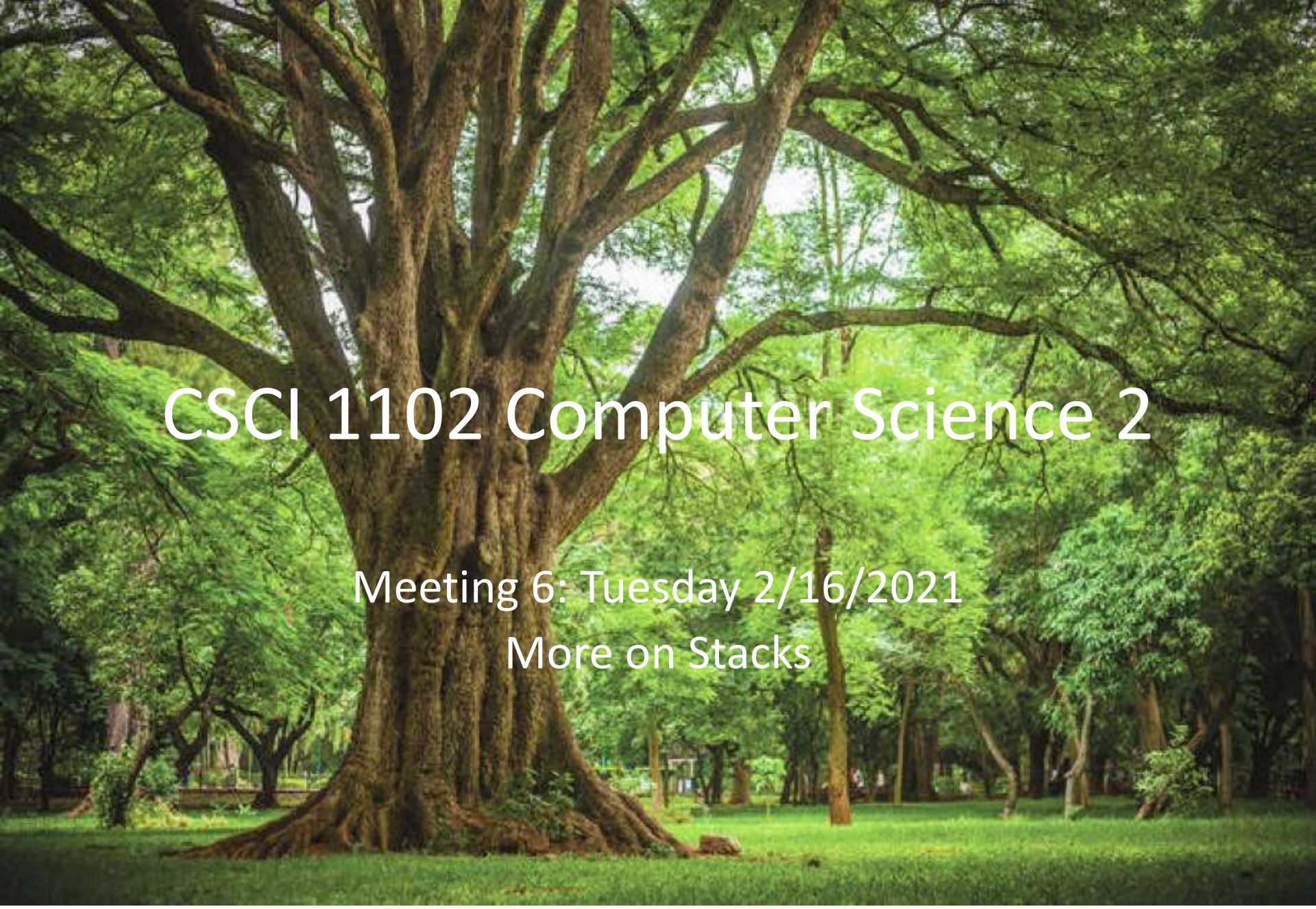


# CSCI 1102 Computer Science 2

Meeting 6: Tuesday 2/16/2021

More on Stacks

```
4   // An API for simple stacks of Strings.
5   //
6   public interface StringStack {
7
8     void push(String s);
9
10    String pop();
11
12    String peek();
13
14    boolean isEmpty();
15
16    String toString();
17  }
```

# StringStack ADT Code

# Improving the StringStack ADT

- Make it polymorphic – stack can house values of any reference type;
- Make it grow and shrink as needed
- A Linked Representation

# Object-Oriented Programming

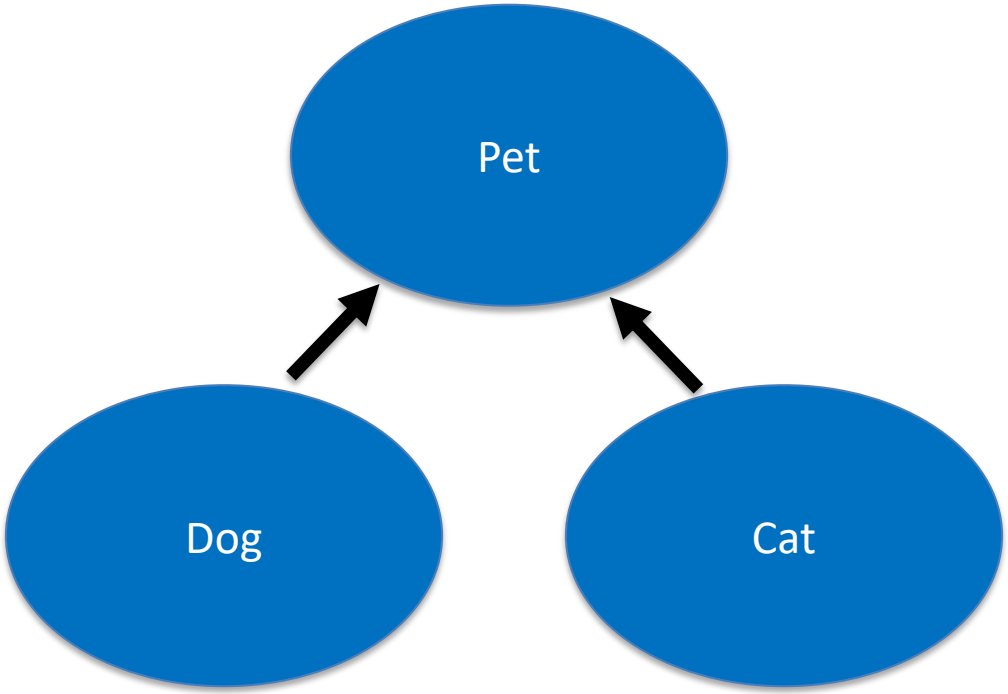
We'll learn enough about it to peacefully co-exist with it.

# Object-Oriented Programming

## Message Passing + Inheritance

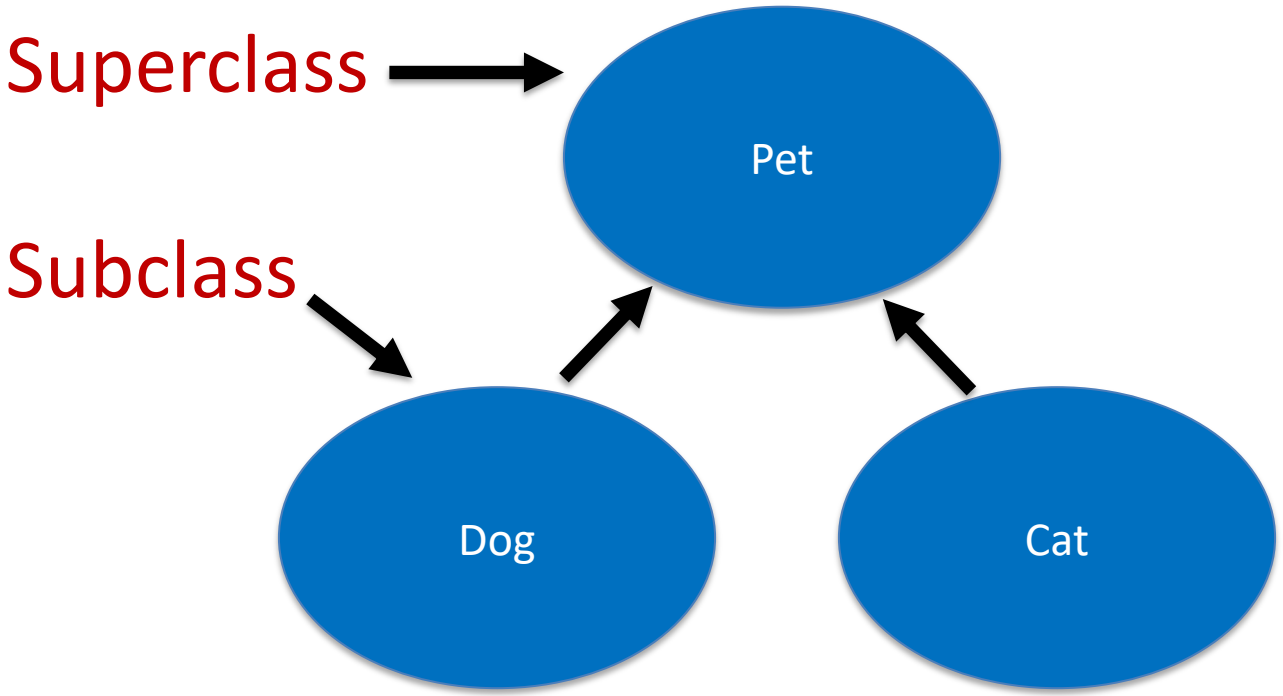
# Inheritance

- In OO, types are organized in an *is-a* hierarchy.

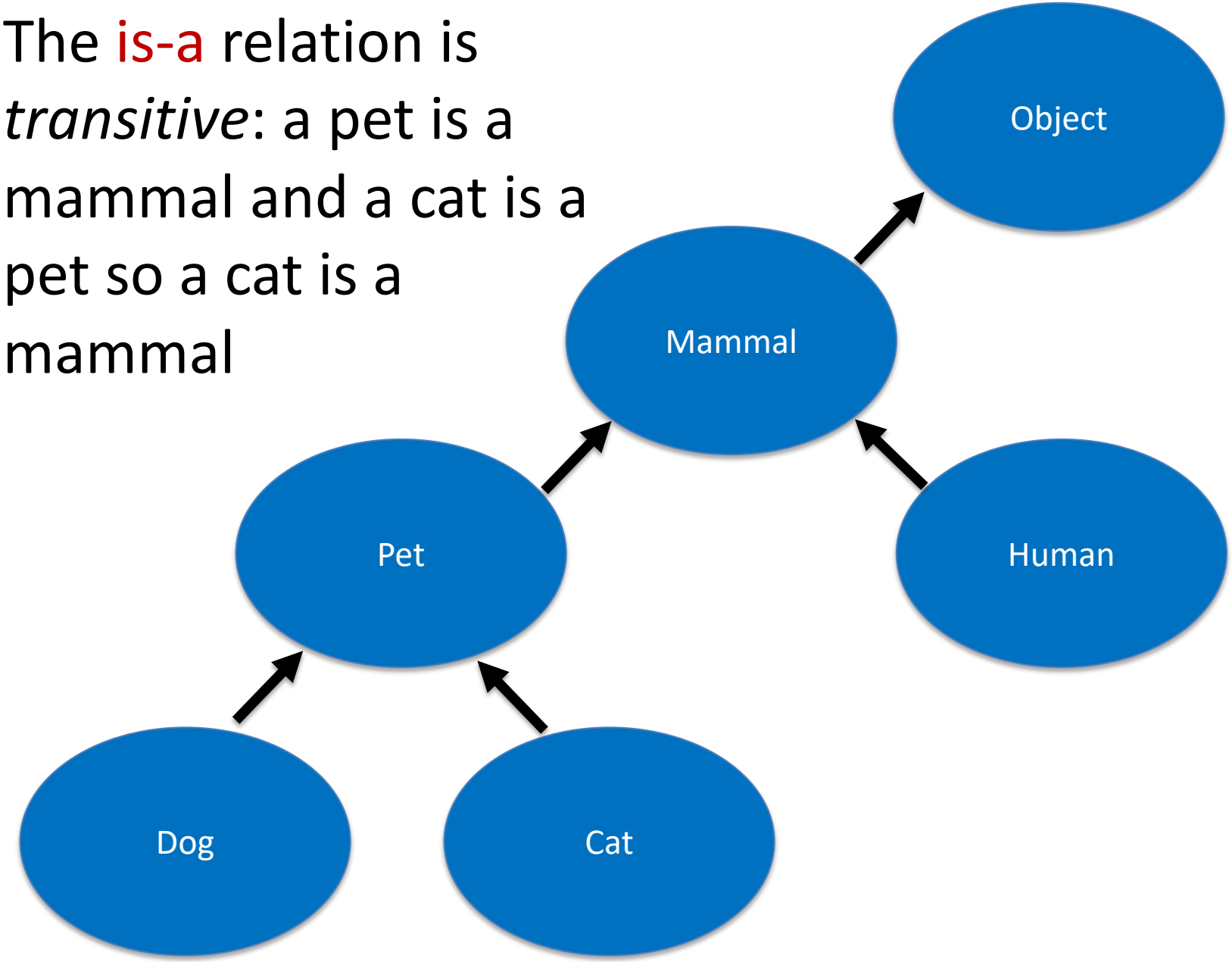


# Inheritance

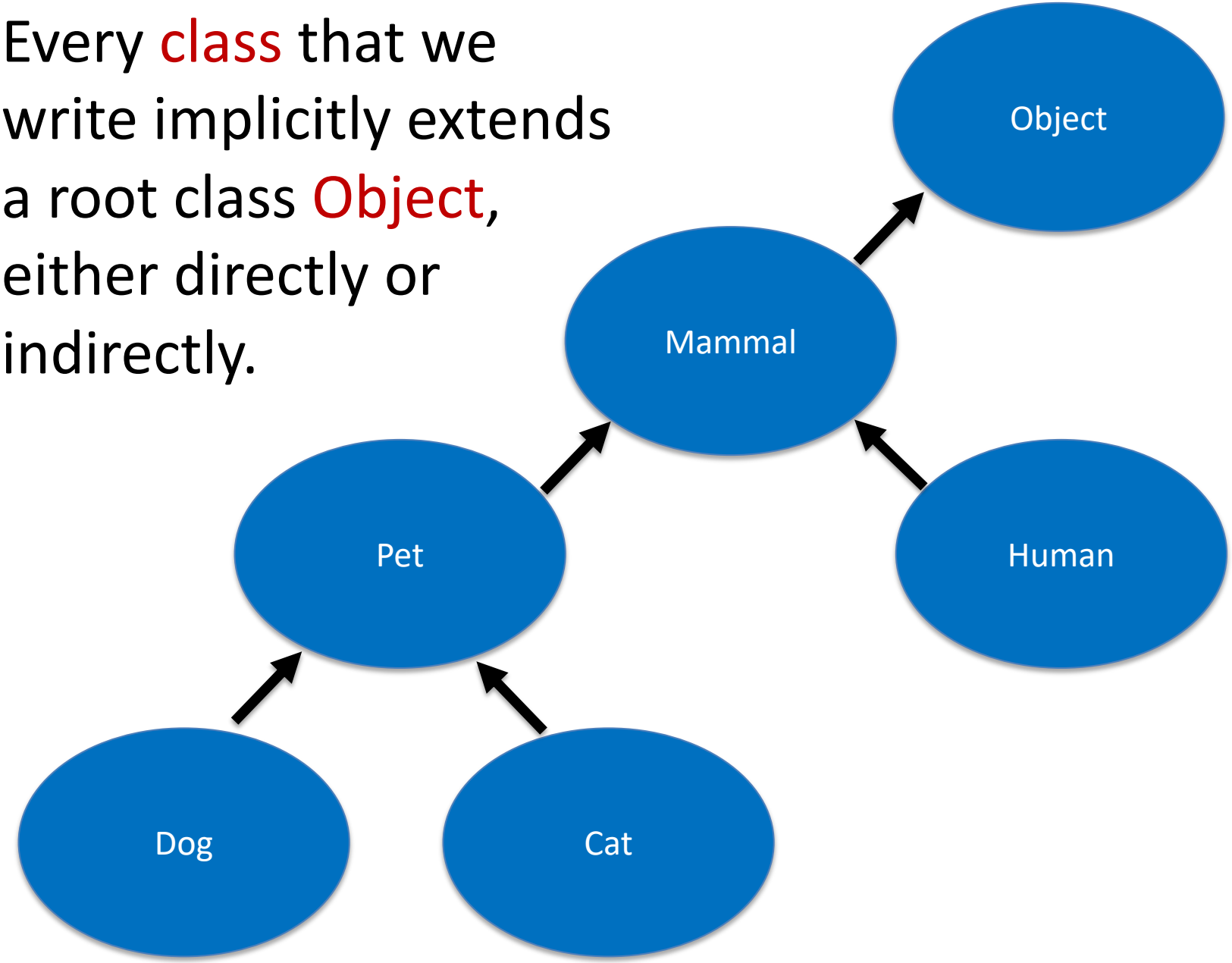
- In OO, types are organized in an *is-a* hierarchy.

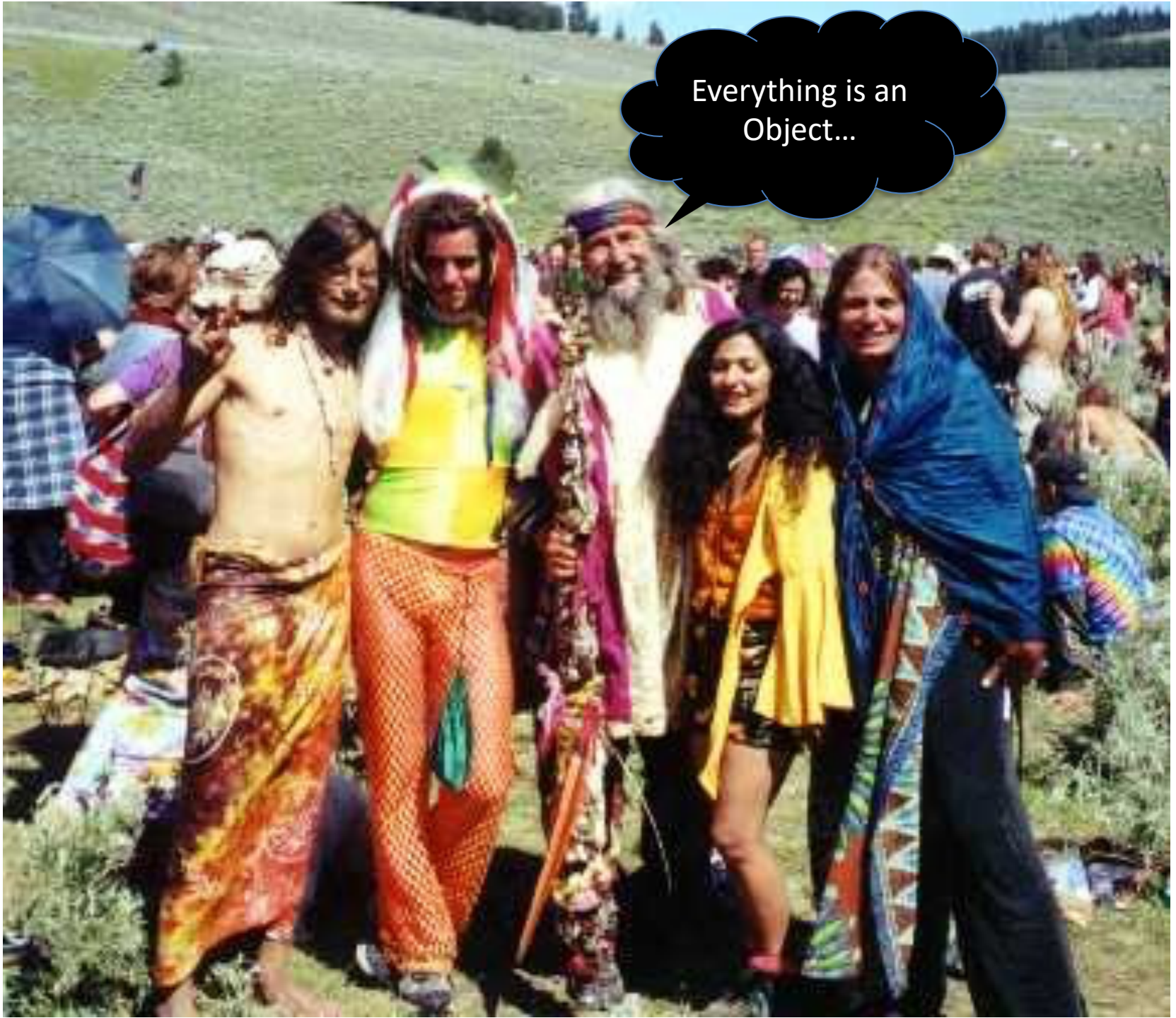


The is-a relation is *transitive*: a pet is a mammal and a cat is a pet so a cat is a mammal



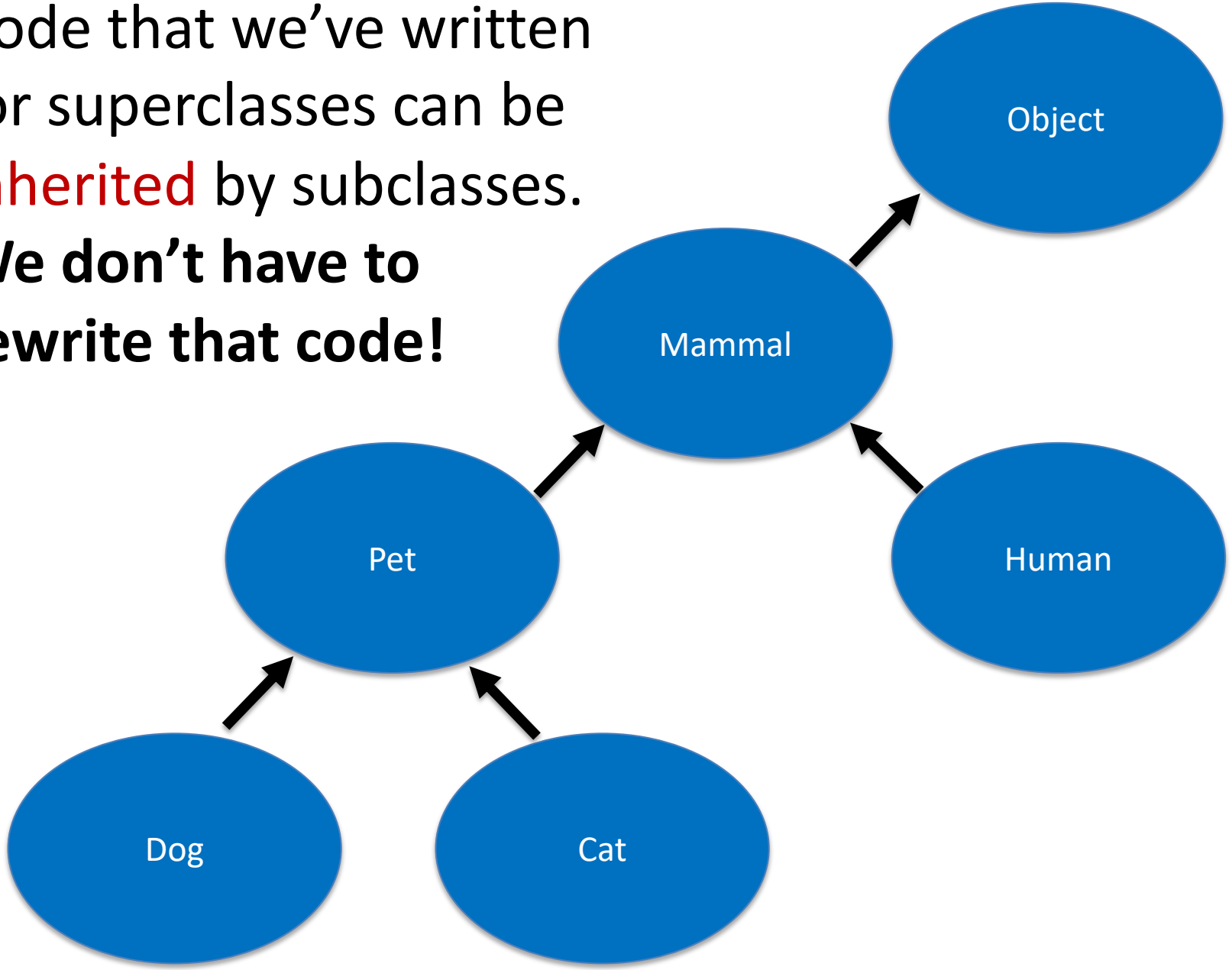
Every class that we write implicitly extends a root class Object, either directly or indirectly.




Everything is an Object…

Code that we've written for superclasses can be inherited by subclasses.

**We don't have to rewrite that code!**



# The MetalComboBoxButton

- org.omg.CORBA.**CompletionStatus** (implements org.omg.CORBA.portable.IDLEntity)
- org.omg.CORBA.**CompletionStatusHelper**
- java.awt.**Component** (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
  - java.awt.**Button** (implements javax.accessibility.Accessible)
  - java.awt.**Canvas** (implements javax.accessibility.Accessible)
  - java.awt.**Checkbox** (implements javax.accessibility.Accessible, java.awt.ItemSelectable)
  - java.awt.**Choice** (implements javax.accessibility.Accessible, java.awt.ItemSelectable)
  - java.awt.**Container**
    - javax.swing.plaf.basic.**BasicSplitPaneDivider** (implements java.beans.PropertyChangeListener)
    - javax.swing.**CellRendererPane** (implements javax.accessibility.Accessible)
    - javax.swing.tree.**DefaultTreeCellEditor.EditorContainer**
    - javax.swing.**JComponent** (implements java.io.Serializable)
      - javax.swing.**AbstractButton** (implements java.awt.ItemSelectable, javax.swing.SwingConstants)
        - javax.swing.**JButton** (implements javax.accessibility.Accessible)
          - javax.swing.plaf.basic.**BasicArrowButton** (implements javax.swing.SwingConstants)
            - javax.swing.plaf.metal.**MetalScrollButton**
          - javax.swing.plaf.metal.**MetalComboBoxButton**
        - javax.swing.**JMenuItem** (implements javax.accessibility.Accessible, javax.swing.MenuElement)
          - javax.swing.**JCheckBoxMenuItem** (implements javax.accessibility.Accessible, javax.swing.SwingC
          - javax.swing.**JMenu** (implements javax.accessibility.Accessible, javax.swing.MenuElement)
          - javax.swing.**JRadioButtonMenuItem** (implements javax.accessibility.Accessible)
        - javax.swing.**JToggleButton** (implements javax.accessibility.Accessible)
          - javax.swing.**JCheckBox** (implements javax.accessibility.Accessible)
          - javax.swing.**JRadioButton** (implements javax.accessibility.Accessible)
      - javax.swing.plaf.basic.**BasicInternalFrameTitlePane**
        - javax.swing.plaf.metal.**MetalInternalFrameTitlePane**
      - javax.swing.**Box** (implements javax.accessibility.Accessible)
      - javax.swing.**Box.Filler** (implements javax.accessibility.Accessible)
      - javax.swing.**JColorChooser** (implements javax.accessibility.Accessible)
      - javax.swing.**JComboBox**<E> (implements javax.accessibility.Accessible, java.awt.event.ActionListener, java.a
        javax.swing.event.ListDataListener)
      - javax.swing.**JFileChooser** (implements javax.accessibility.Accessible)
      - javax.swing.**JInternalFrame** (implements javax.accessibility.Accessible, javax.swing.RootPaneContainer, java
      - javax.swing.**JInternalFrame.JDesktopIcon** (implements javax.accessibility.Accessible)
      - javax.swing.**JLabel** (implements javax.accessibility.Accessible, javax.swing.SwingConstants)
        - javax.swing.plaf.basic.**BasicComboBoxRenderer** (implements javax.swing.ListCellRenderer<E>, java.i

Pet:

```
String getOwner() {…}
String getName() {…}
double getWeight() {…}
String says() {…}
```

Dog:

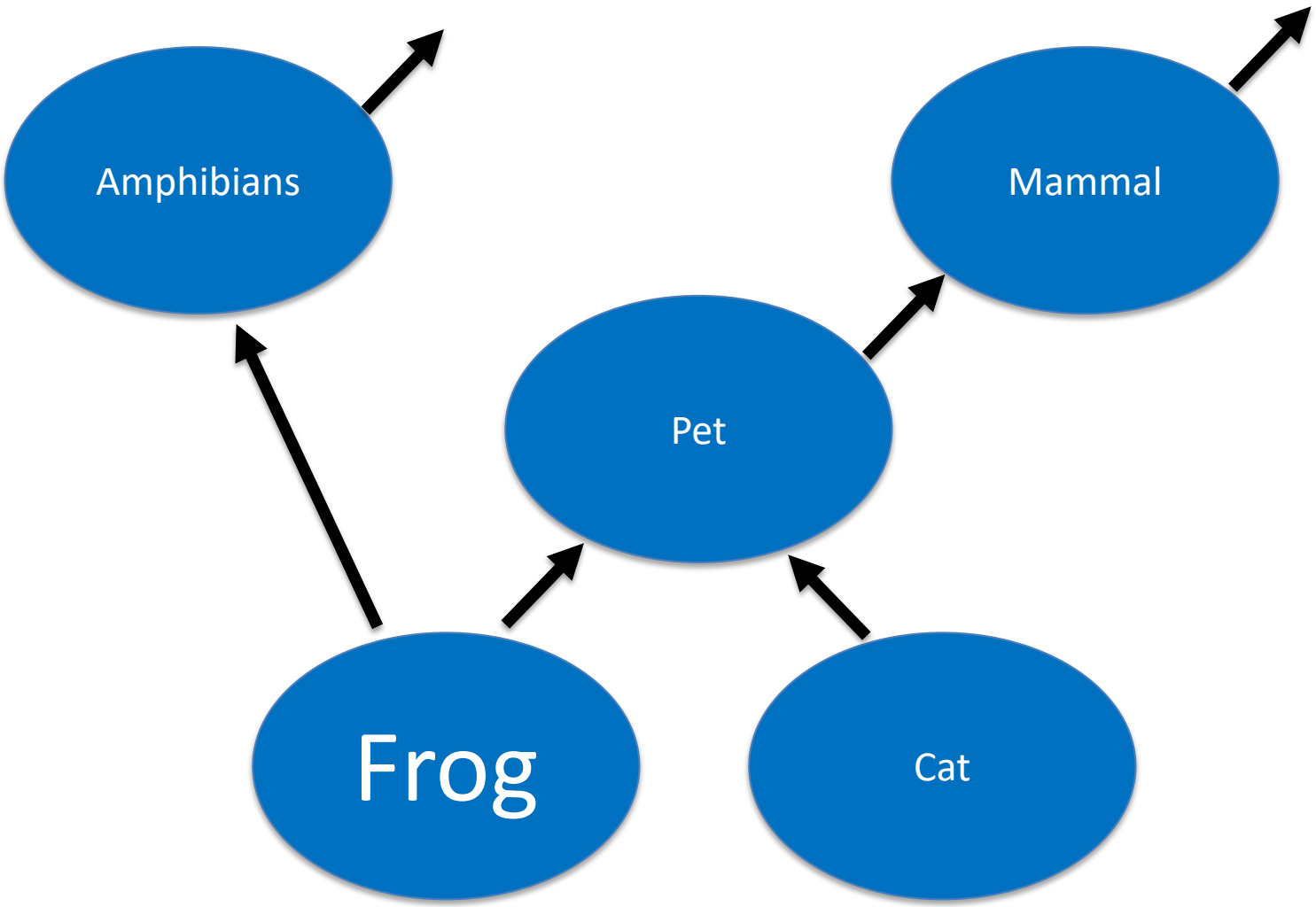
```
@Override
String says() {
  return “Woof”;
}
```

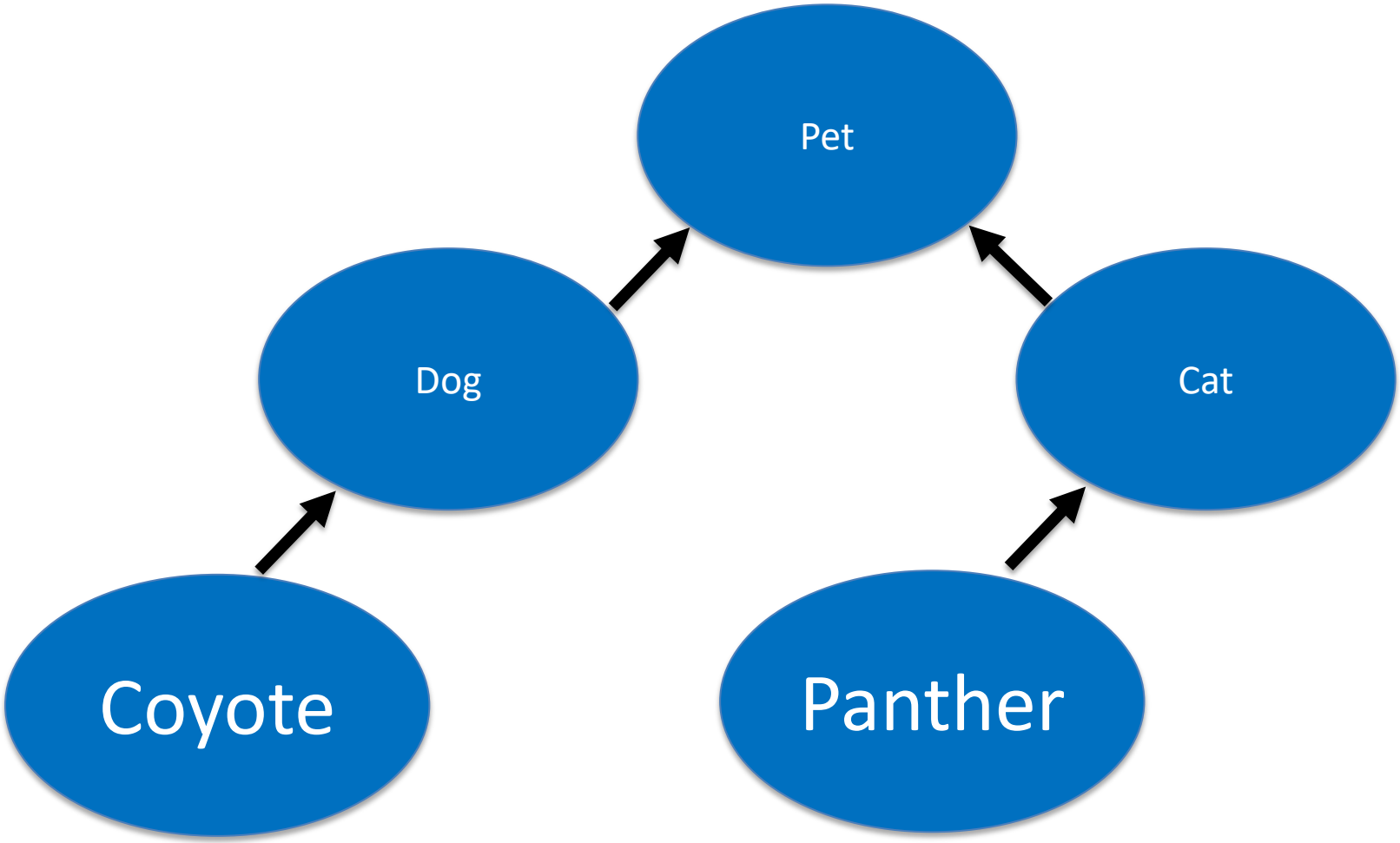
getOwner, getName, getWeight are inherited by Dog. Dog overrides the says function.

# Three Problems

# Problem 1: Category Errors

# Not all Pets are Mammals.



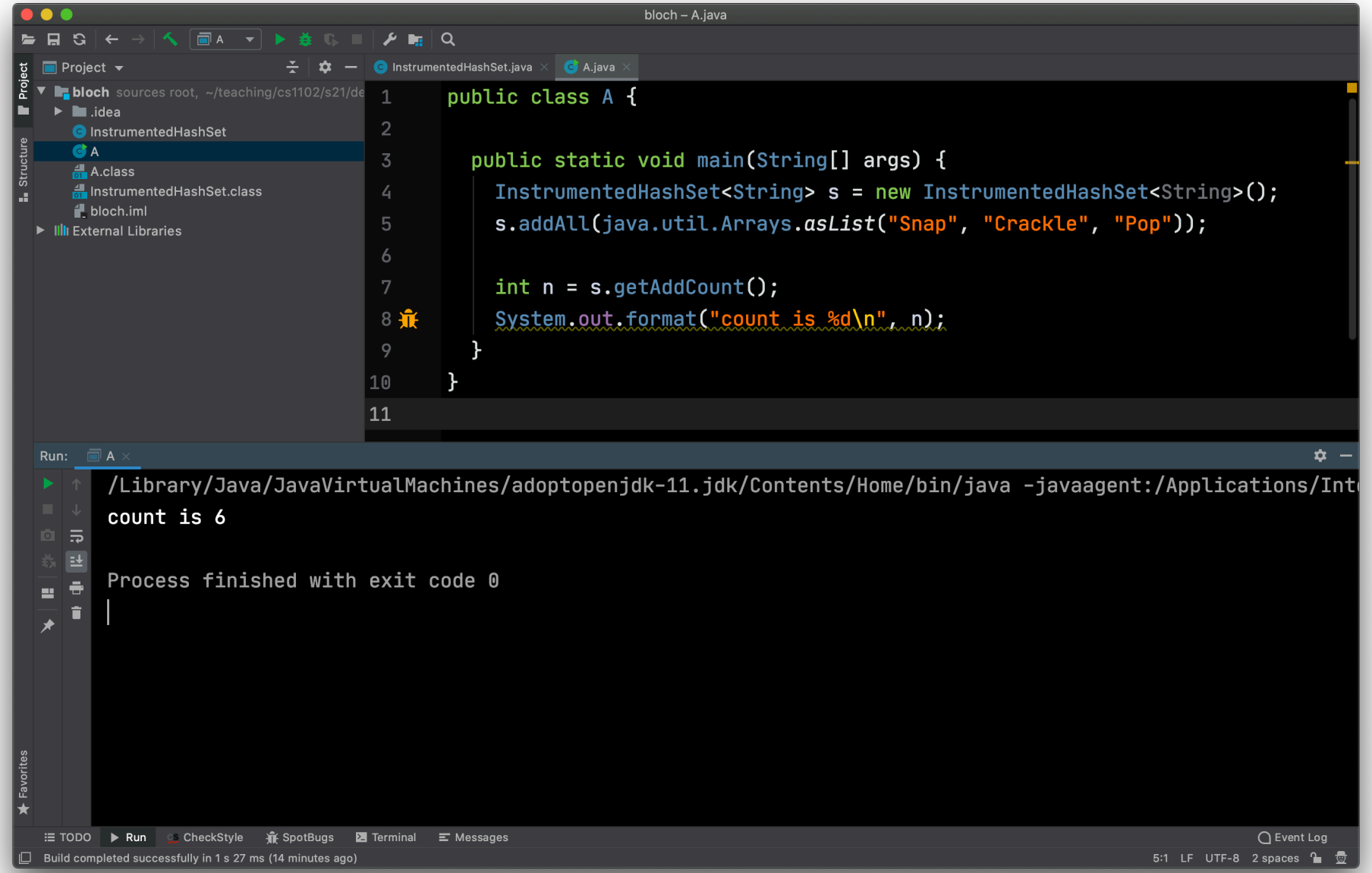

Pet
Dog
Cat
Coyote
Panther

# is-a Hierarchies

- Real life systems are complicated and generally don't conform to simple, transitive is-a hierarchies;
- This leads to the development of artificially complex inheritance hierarchies;
- The code base becomes brittle, hard to maintain, hard to extend, easy to break.

# Problem 2: Inheritance isn’t modular

```java
// Broken - Inappropriate use of inheritance!
public class InstrumentedHashSet<E> extends java.util.HashSet<E> {
  // The number of attempted element insertions
  private int addCount;

  public InstrumentedHashSet() {
    this.addCount = 0;
  }

  @Override
  public boolean add(E e) {
    this.addCount++;
    return super.add(e);
  }

  @Override
  public boolean addAll(java.util.Collection<? extends E> c) {
    this.addCount += c.size();
    return super.addAll(c);
  }
```


bloch – A.java
Project
bloch sources root, ~/teaching/cs1102/s21/de
.idea
InstrumentedHashSet
A
A.class
InstrumentedHashSet.class
bloch.iml
External Libraries
InstrumentedHashSet.java
A.java
1 public class A {
2
3 public static void main(String[] args) {
4 InstrumentedHashSet<String> s = new InstrumentedHashSet<String>();
5 s.addAll(java.util.Arrays.asList("Snap", "Crackle", "Pop"));
6
7 int n = s.getAddCount();
8 System.out.format("count is %d\n", n);
9 }
10 }
11
Run: A
/Library/Java/JavaVirtualMachines/adoptopenjdk-11.jdk/Contents/Home/bin/java -javaagent:/Applications/Int
count is 6
Process finished with exit code 0
TODO
Run
CheckStyle
SpotBugs
Terminal
Messages
Event Log
Build completed successfully in 1 s 27 ms (14 minutes ago)
5:1 LF UTF-8 2 spaces
Favorites

# Problem 3: Inheritance is slow.

SUMMARY: NESTED | FIELD | CONSTR | METHOD  DETAIL: FIELD | CONSTR | METHOD

compact1, compact2, compact3
java.lang

## Class Object

java.lang.Object

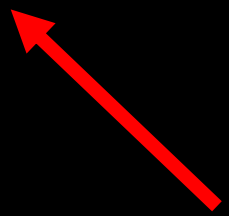
### *Method Summary*

**All Methods** | **Instance Methods** | **Concrete Methods**

| Modifier and Type | Method and Description |
|---|---|
| protected Object | **clone**()<br>Creates and returns a copy of this object. |
| boolean | **equals**(Object obj)<br>Indicates whether some other object is "equal to" this one. |
| protected void | **finalize**()<br>Called by the garbage collector on an object when garbage collection determines that there are no more references to the object. |
| Class<?> | **getClass**()<br>Returns the runtime class of this Object. |
| int | **hashCode**()<br>Returns a hash code value for the object. |
| void | **notify**()<br>Wakes up a single thread that is waiting on this object's monitor. |
| void | **notifyAll**()<br>Wakes up all threads that are waiting on this object's monitor. |
| String | **toString**()<br>Returns a string representation of the object. |

```java
// file: ObjectStack.java
// author: Bob Muller
//
// An API for simple stacks of Objects.
//
public interface ObjectStack {

  void push(Object s);

  Object pop();

  Object peek();

  boolean isEmpty();

  String toString();
}
```

# Pre-Generic Style – this is obsolete but you'll see it.

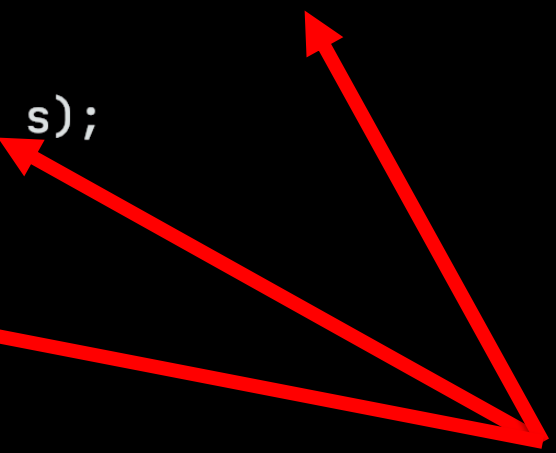


Since everything is an Object, the stack can house anything.

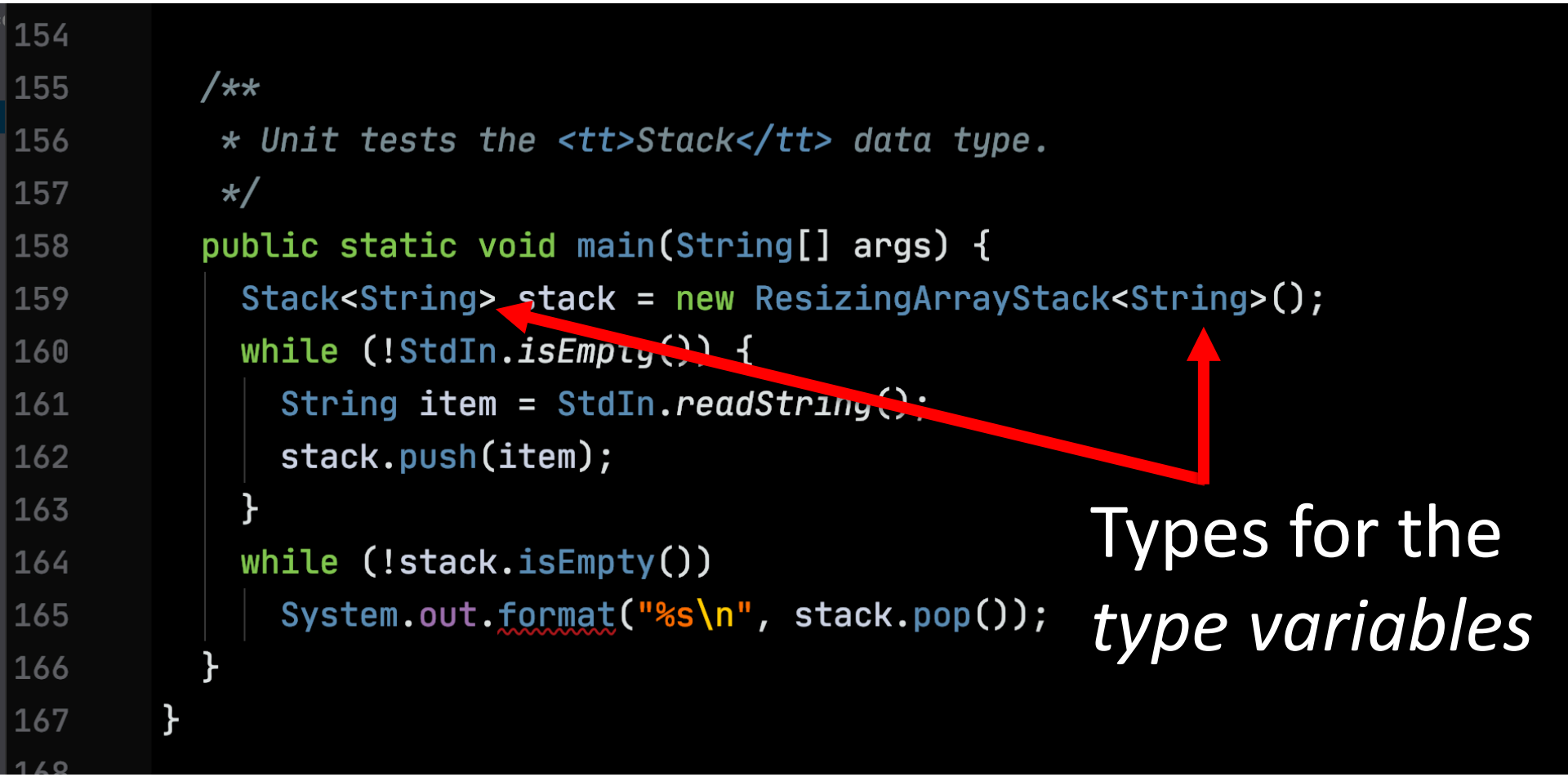
```
15    private Object[] theStack;
16
17    public ObjectStackC() {
18      this.top = 0;
19      this.theStack = new Object[CAPACITY];
20    }
21
22    public void push(Object s) {
23      if (this.top == CAPACITY)
24        throw new RuntimeException("Stack Overflow");
25      else
26        this.theStack[this.top++] = s;
27    }
28
29    public Object pop() {
30      if (this.top == 0)
31        throw new NoSuchElementException("Stack Underflow");
32      else {
33        Object item = this.theStack[--this.top];
34        this.theStack[this.top] = null;
35        return item;
36      }
37    }
```

```
61
62    public static void main(String[] args) {
63
64      // Unit testing
65      ObjectStack myStack = new ObjectStackC();
66
67      myStack.push("Alice");
68      myStack.push("Jose");
69      myStack.push("Bob");
70
71      while (!myStack.isEmpty()) {
72        String item = (String) myStack.pop();
73        System.out.format("%s\n", item);
74      }
75    }
```

```
5   //
6   public interface Stack<T> {
7
8     void push(T s);
9
10    T pop();
11
12    T peek();
13
14    boolean isEmpty();
15
16    int size();
17
18    String toString();
19  }
20
```



Generic *type variables*

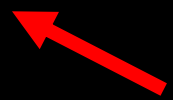

154
155 /**
156 * Unit tests the <tt>Stack</tt> data type.
157 */
158 public static void main(String[] args) {
159 Stack<String> stack = new ResizingArrayStack<String>();
160 while (!StdIn.isEmpty()) {
161 String item = StdIn.readString();
162 stack.push(item);
163 }
164 while (!stack.isEmpty())
165 System.out.format("%s\n", stack.pop());
166 }
167 }
Types for the
type variables

# Resizing

```
73
74      // resize the underlying array holding the elements
75      private void resize(int capacity) {
76        assert capacity >= N;
77        @SuppressWarnings("unchecked")
78        Item[] temp = (Item[]) new Object[capacity];
79        for (int i = 0; i < N; i++) {
80          temp[i] = a[i];
81        }
82        a = temp;
83      }
84
85      /**
86       * Adds the item to this stack.
87       *
88       * @param item the item to add
89       */
90      public void push(Item item) {
91        if (N == a.length) resize(2 * a.length);    // double size of array if nec
92        a[N++] = item;                              // add item
93      }
```

Storage for a now free

```
100     */
101    public Item pop() {
102       if (isEmpty()) throw new NoSuchElementException("Stack underflow");
103       Item item = a[N - 1];
104       a[N - 1] = null;                              // to avoid loitering
105       N--;
106       // shrink size of array if necessary
107       if (N > 0 && N == a.length / 4) resize(a.length / 2);
108       return item;
109    }
110
```



Only using ¼, free it

Pet:
String getOwner()
String getName()
double getWeight()
String says()

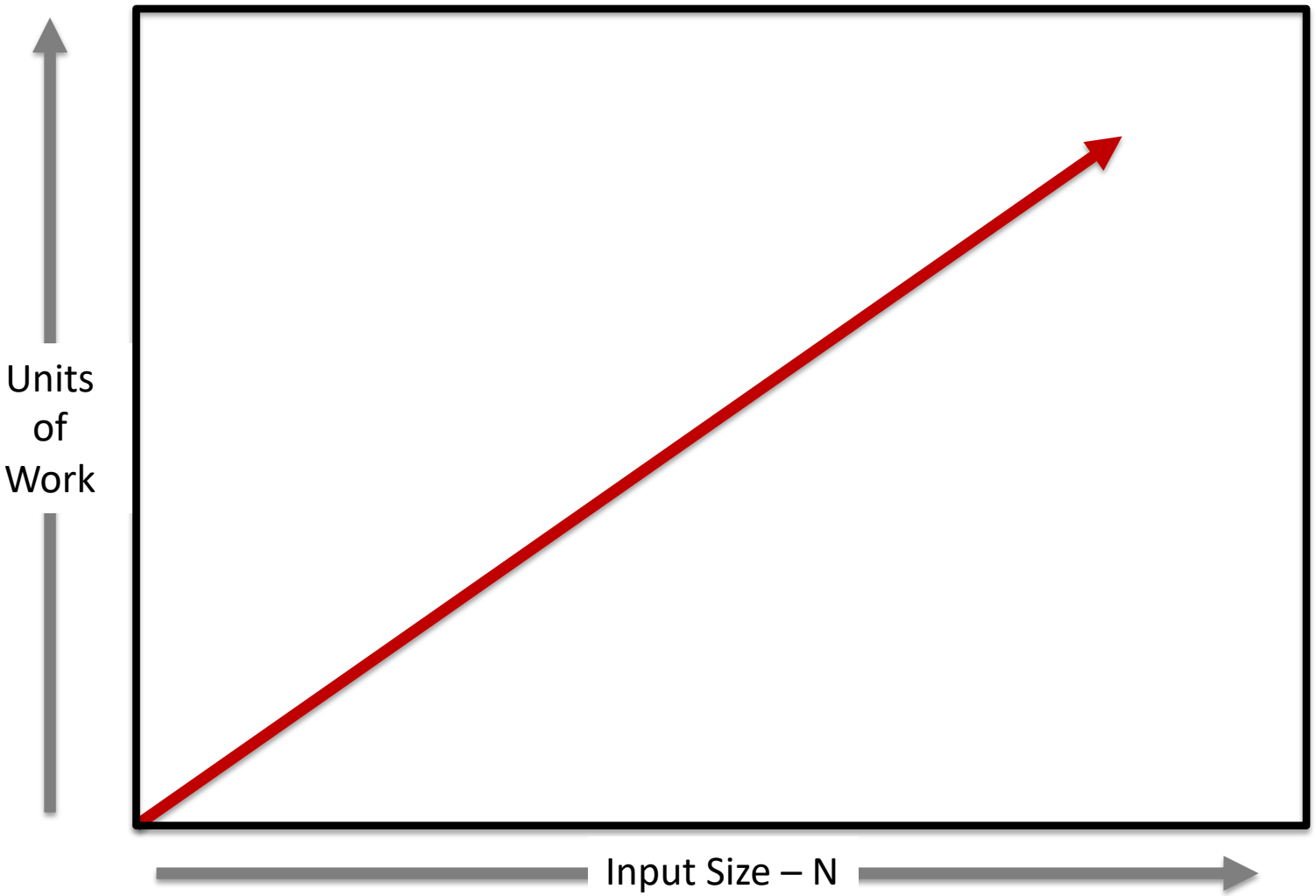
# A Linked Implementation of the StringStack ADT

# Work

# How Much?

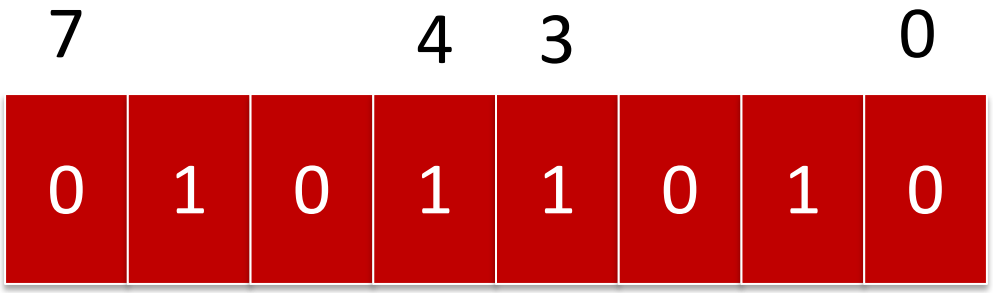
- How much time does an operation take?
- How much memory does it use?
- In the worst case? On average?
- How does the work grow as the input grows?

# Measure the Input Size – N

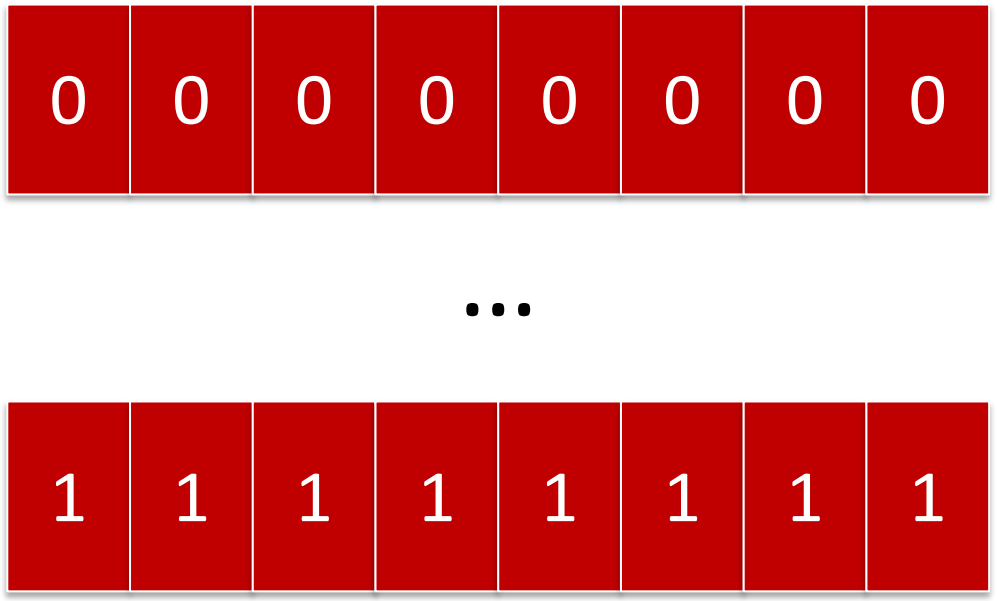


# Memory

# Basic unit of storage is the byte



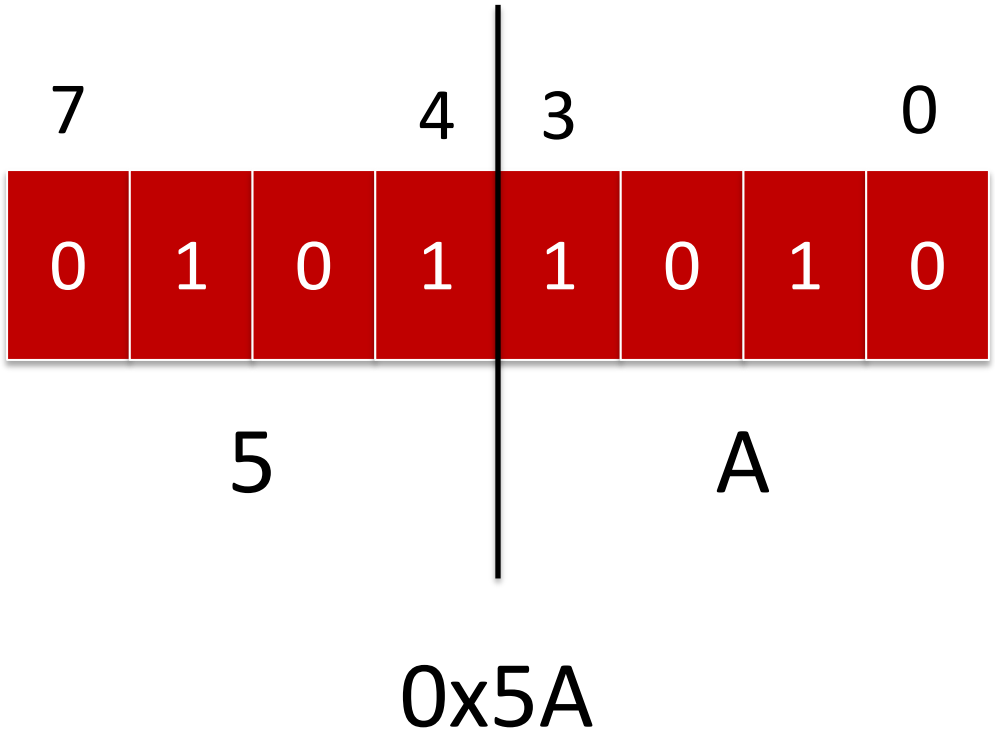
# There are $2^8$ = 256 8-bit patterns



# We can abbreviate 4 bits with one hexadecimal (base 16) digit

| Binary | Hex | Binary | Hex |
|---|---|---|---|
| 0000 | 0 | 1000 | 8 |
| 0001 | 1 | 1001 | 9 |
| 0010 | 2 | 1010 | A |
| 0011 | 3 | 1011 | B |
| 0100 | 4 | 1100 | C |
| 0101 | 5 | 1101 | D |
| 0110 | 6 | 1110 | E |
| 0111 | 7 | 1111 | F |

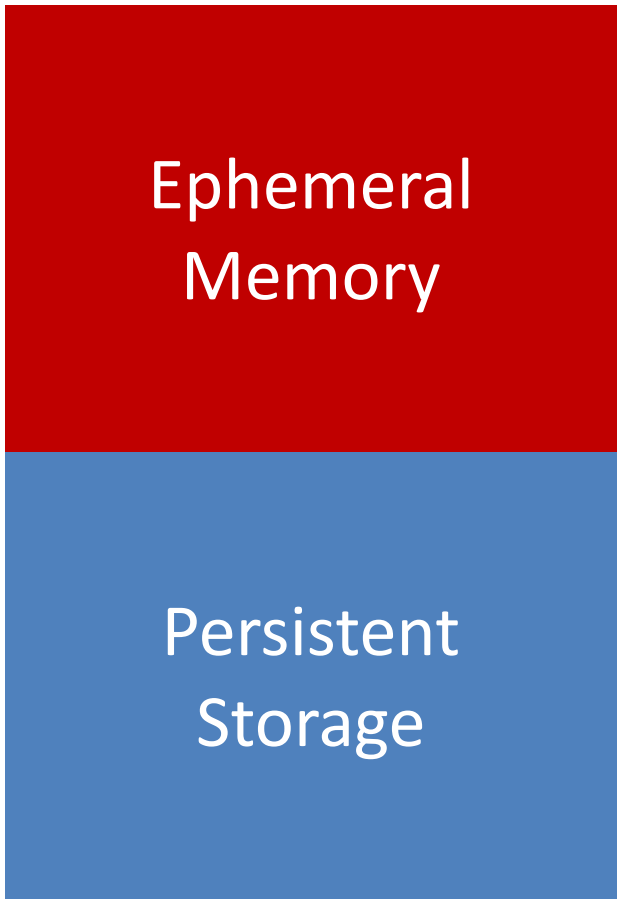
# Abbreviate an 8-bit byte with a pair of hex digits



# Memory Organization

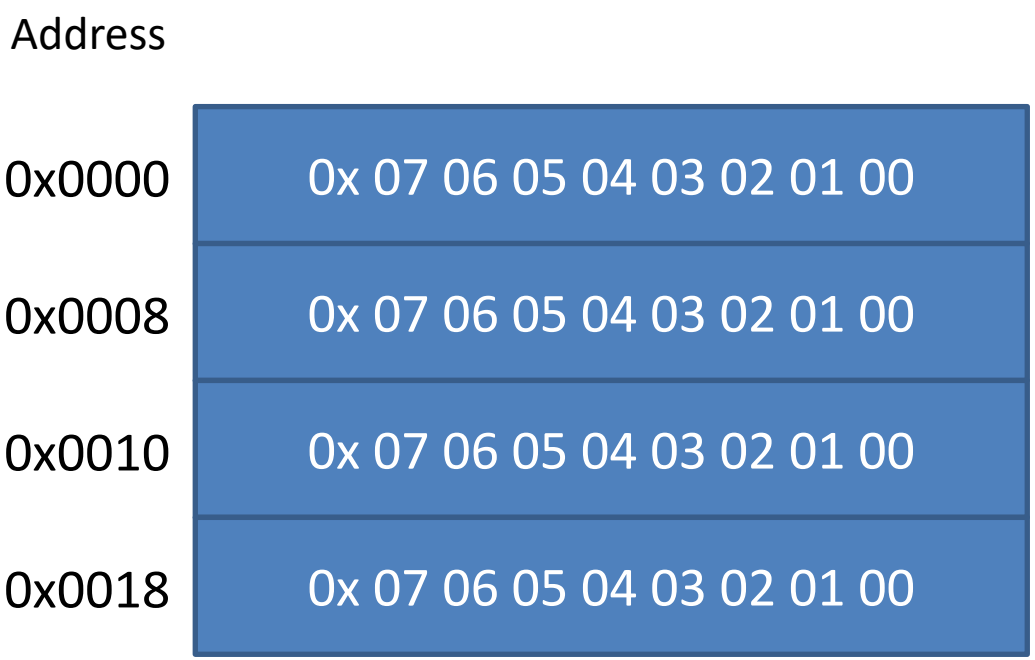
Mostly Random Access Memory (RAM)

Flash Drive

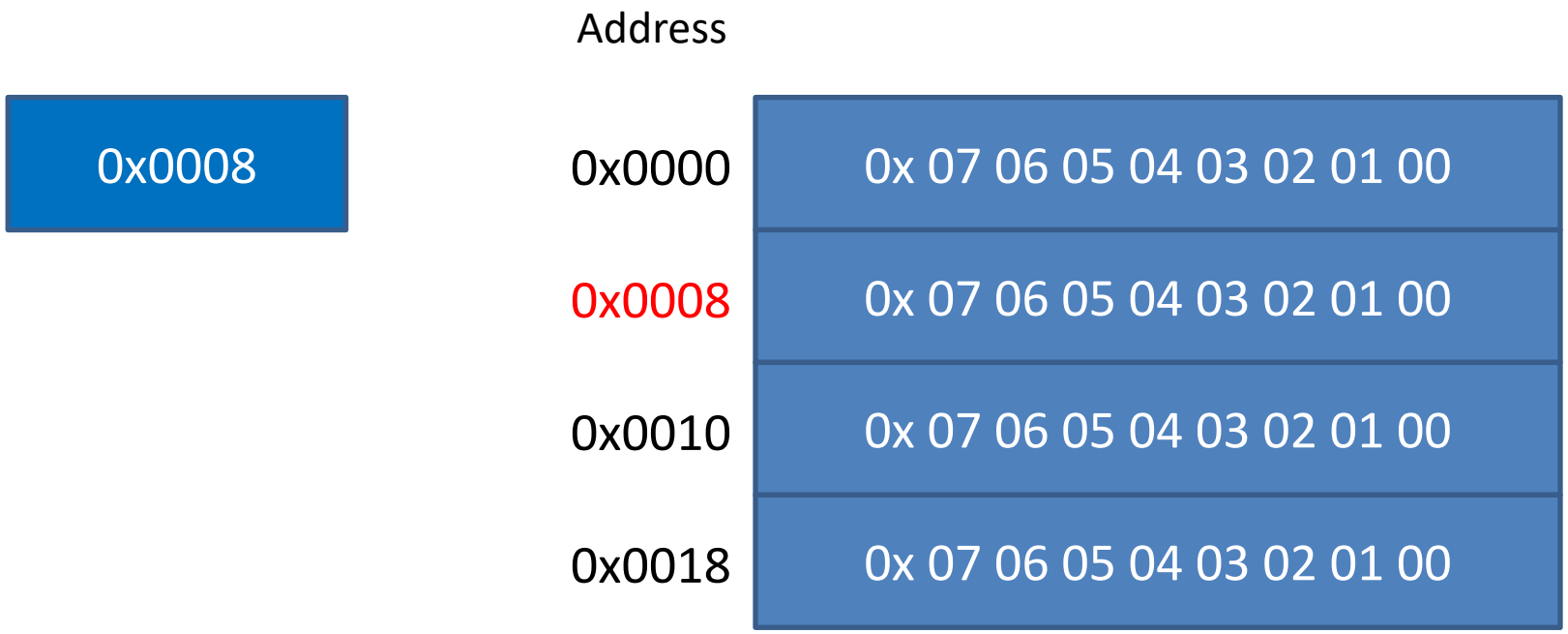


# RAM: Contiguously allocated bytes – each byte has a numerical address

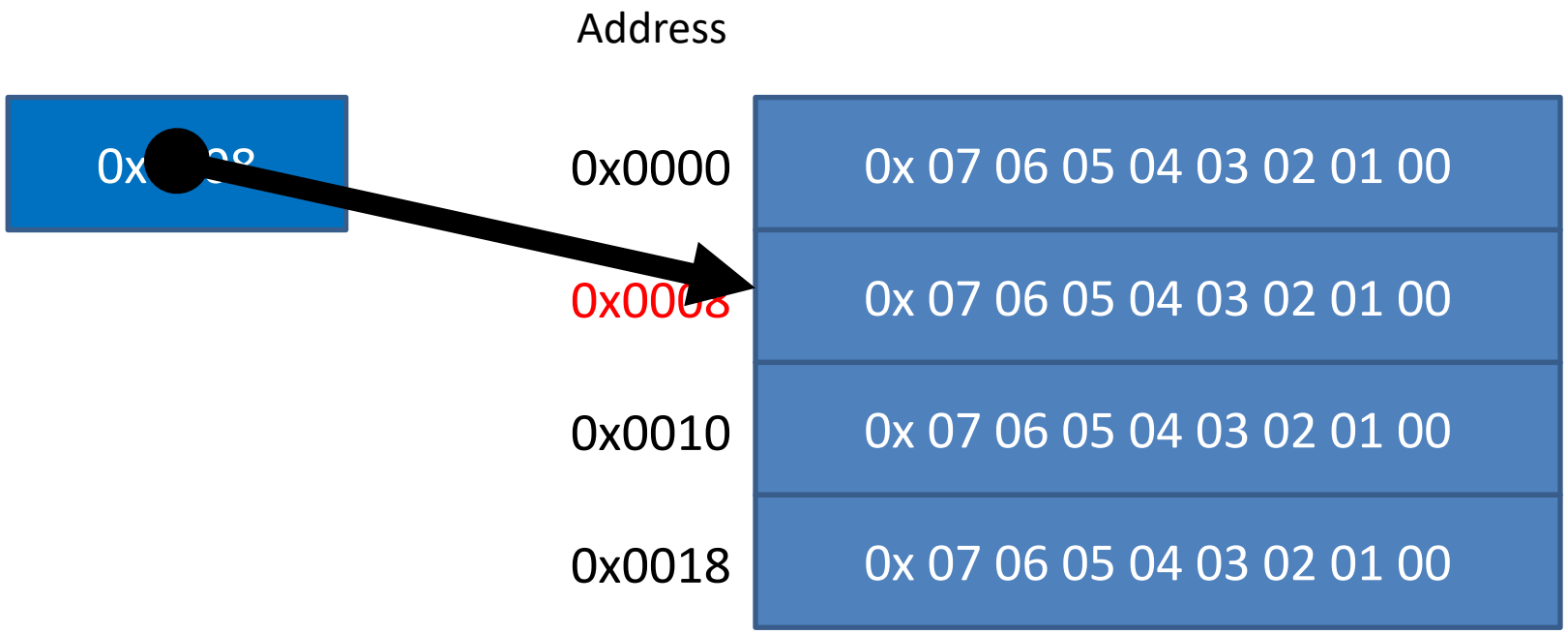
# Words – 8 consecutive bytes



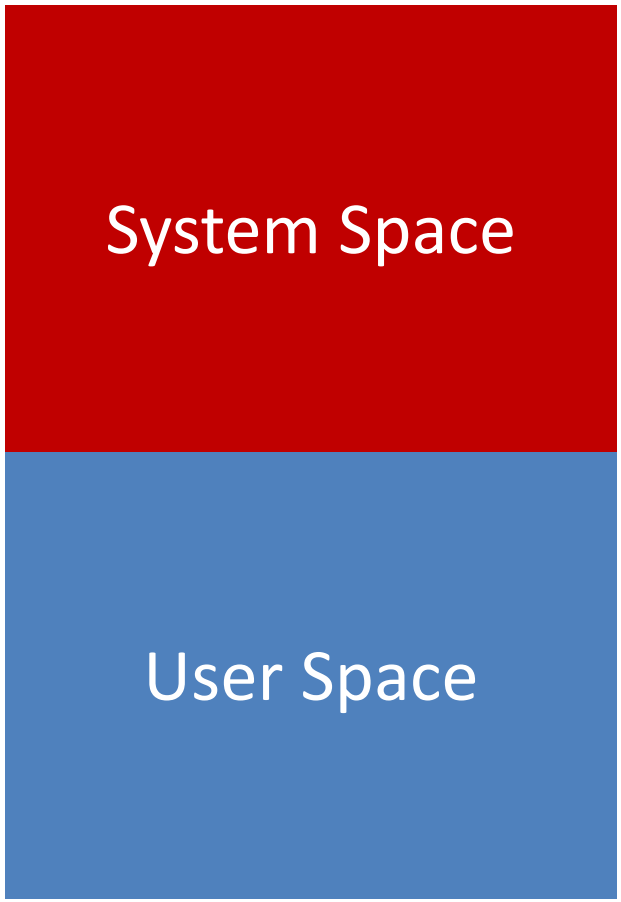
# Variables often hold Addresses



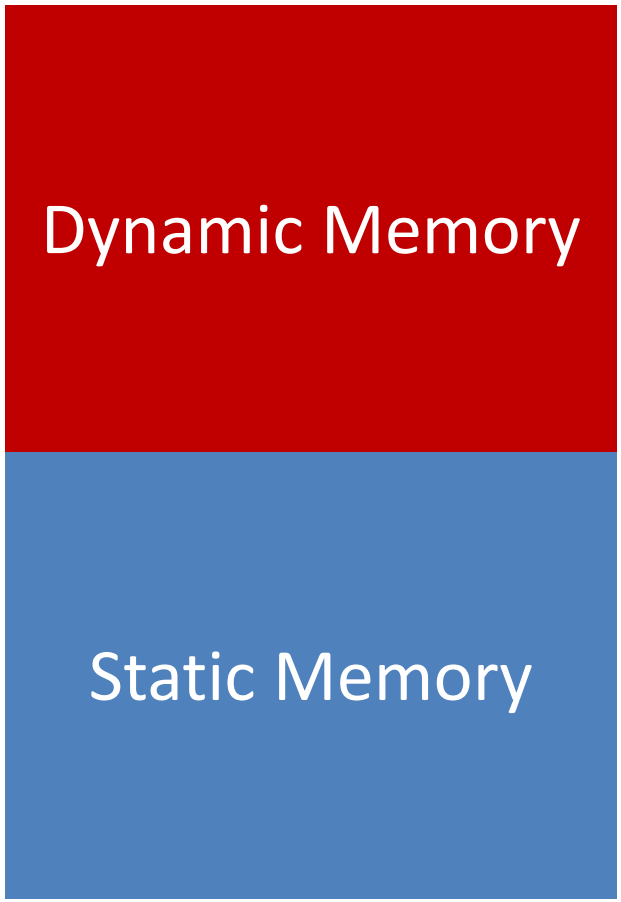
# In diagrams, addresses are usually depicted abstractly as arrows

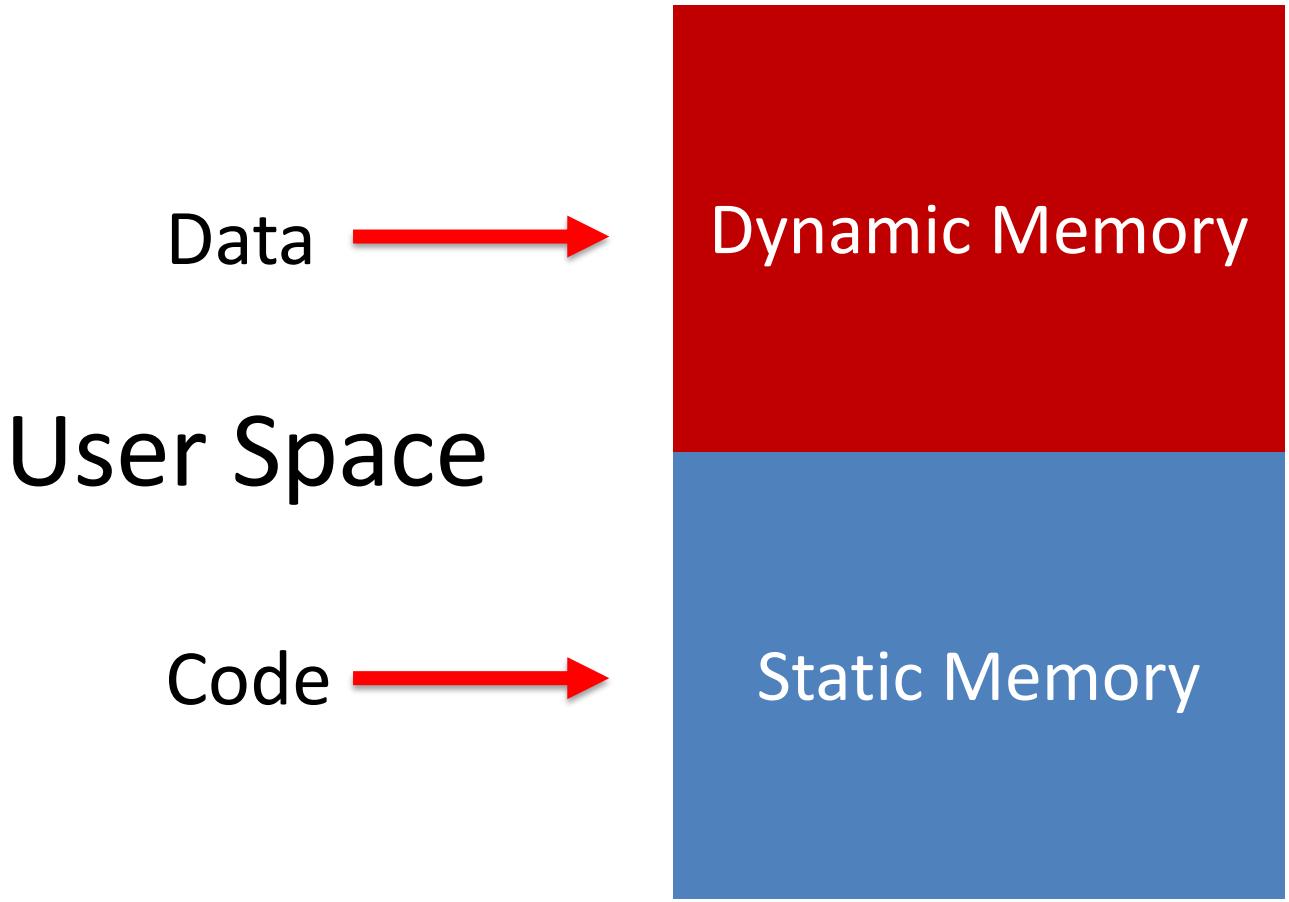


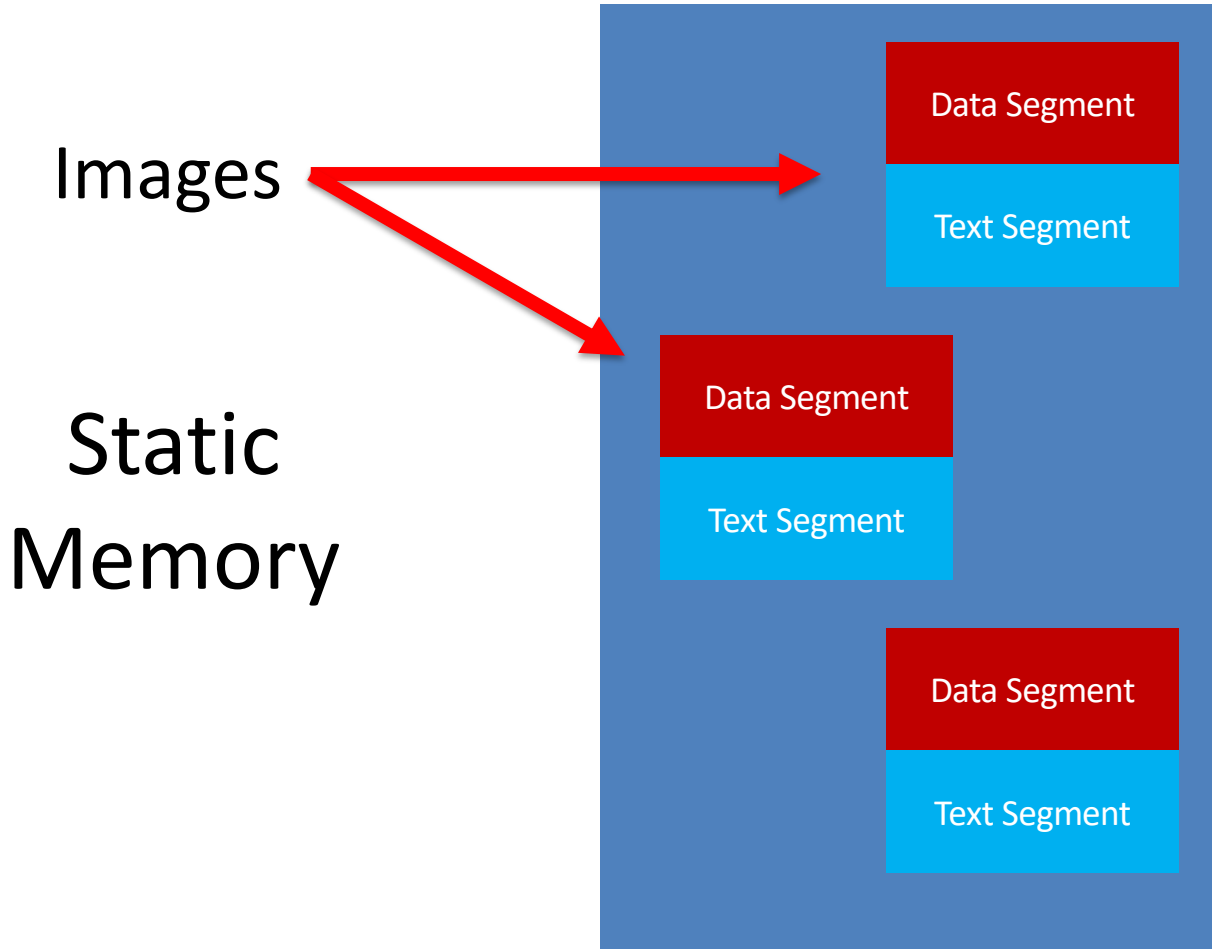
Ephemeral Memory



User Space

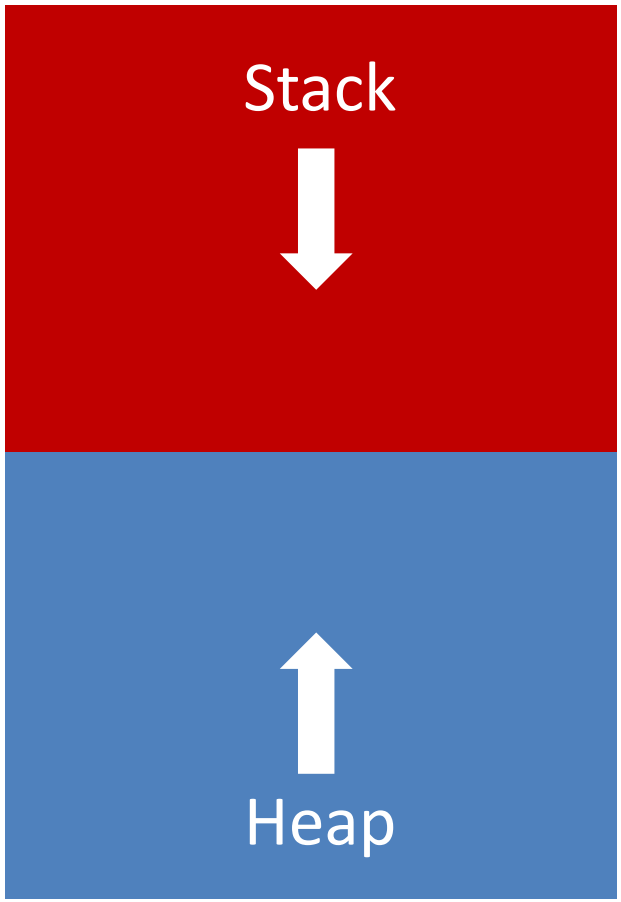



Data
Dynamic Memory
User Space
Code
Static Memory

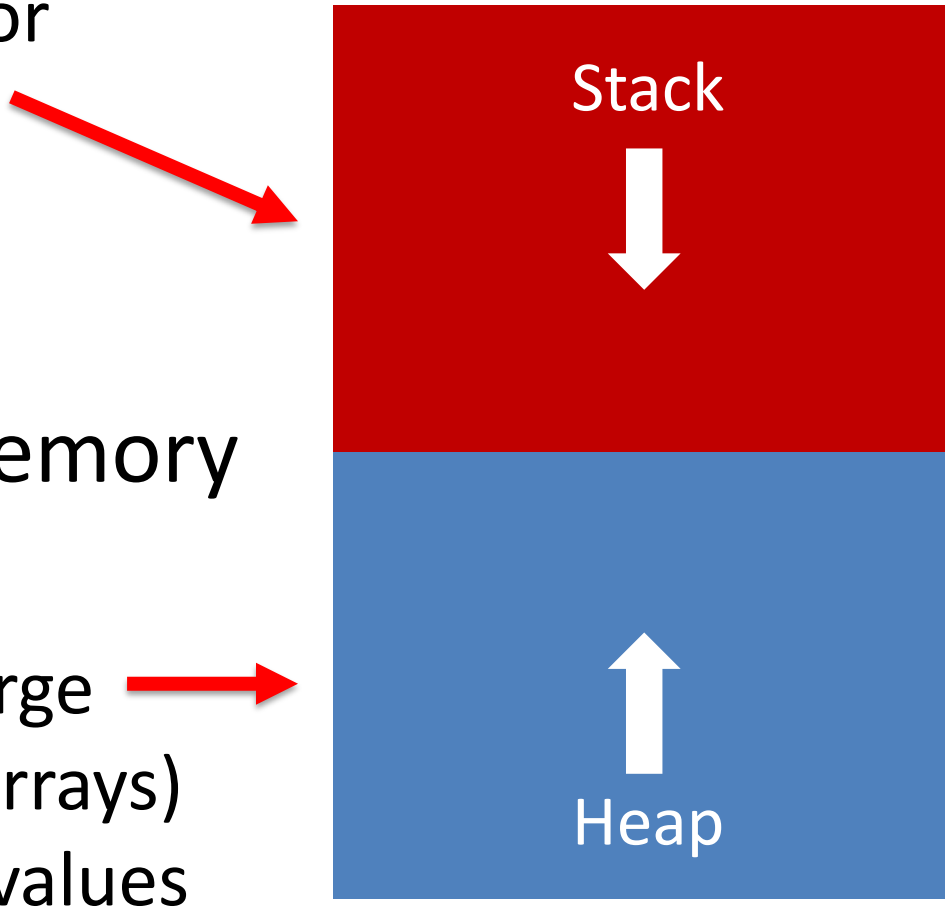

Images
Static Memory
Data Segment
Text Segment
Data Segment
Text Segment
Data Segment
Text Segment

**All** function/method definitions can be understood as images residing in static memory

# Dynamic Memory



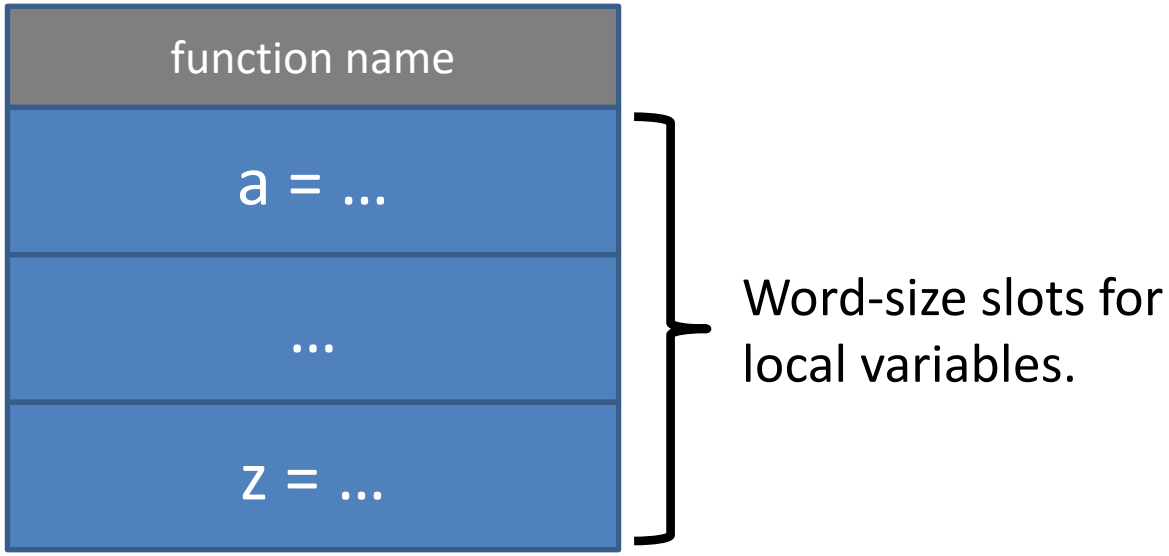
Storage for function variables




# Dynamic Memory

Storage for large values (e.g., arrays) & long-living values

# The Call Stack: Call Frames/Activation Records




Body of function evaluated with respect to a call frame.

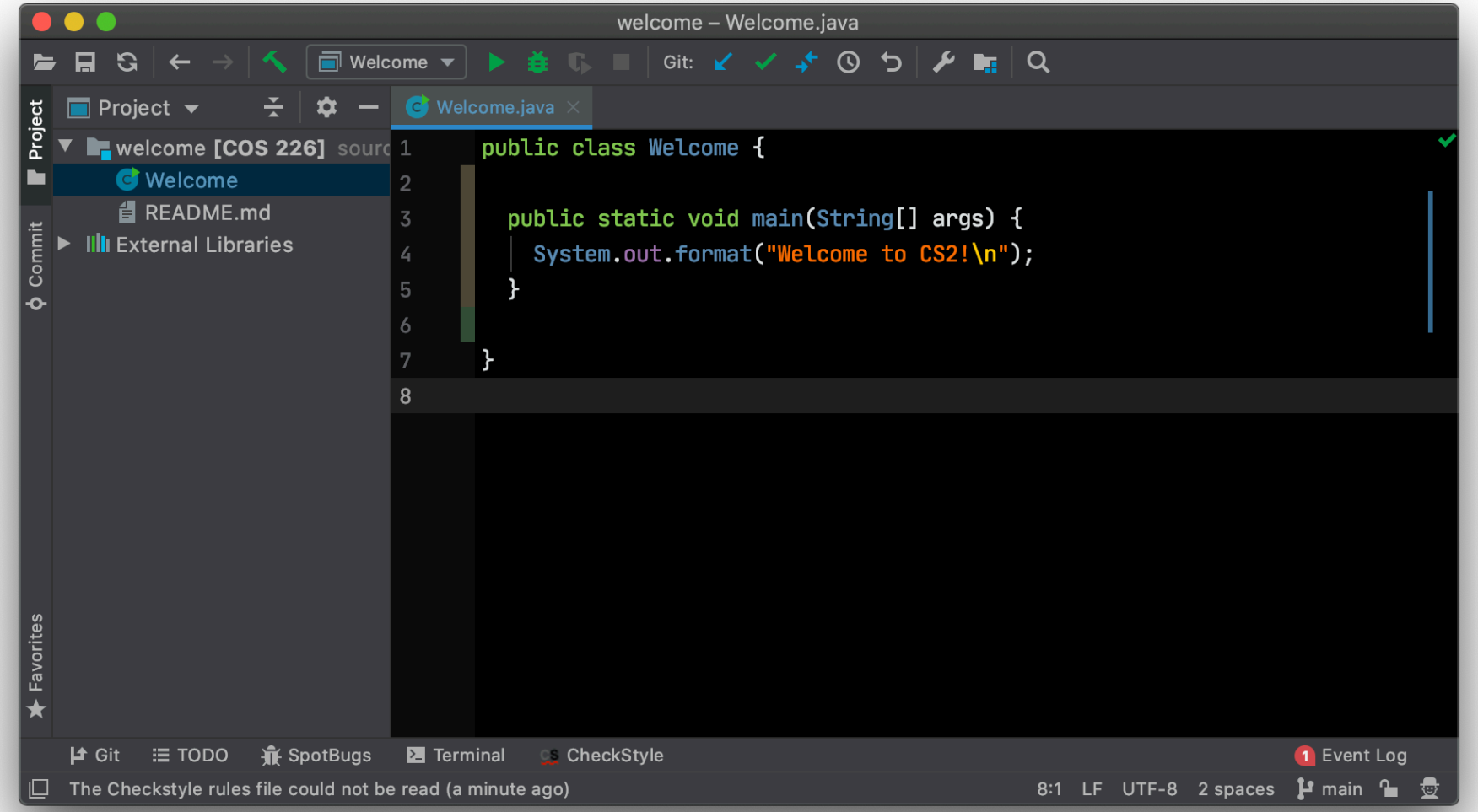
# Managing Dynamic Memory

- The stack: the compiler generates code that manages the allocation and deallocation of call frames on the call stack

- A call frame is placed on top of the stack on function call and removed on function return

- The heap: managed by a run-time support routine called a garbage collector.

# Example

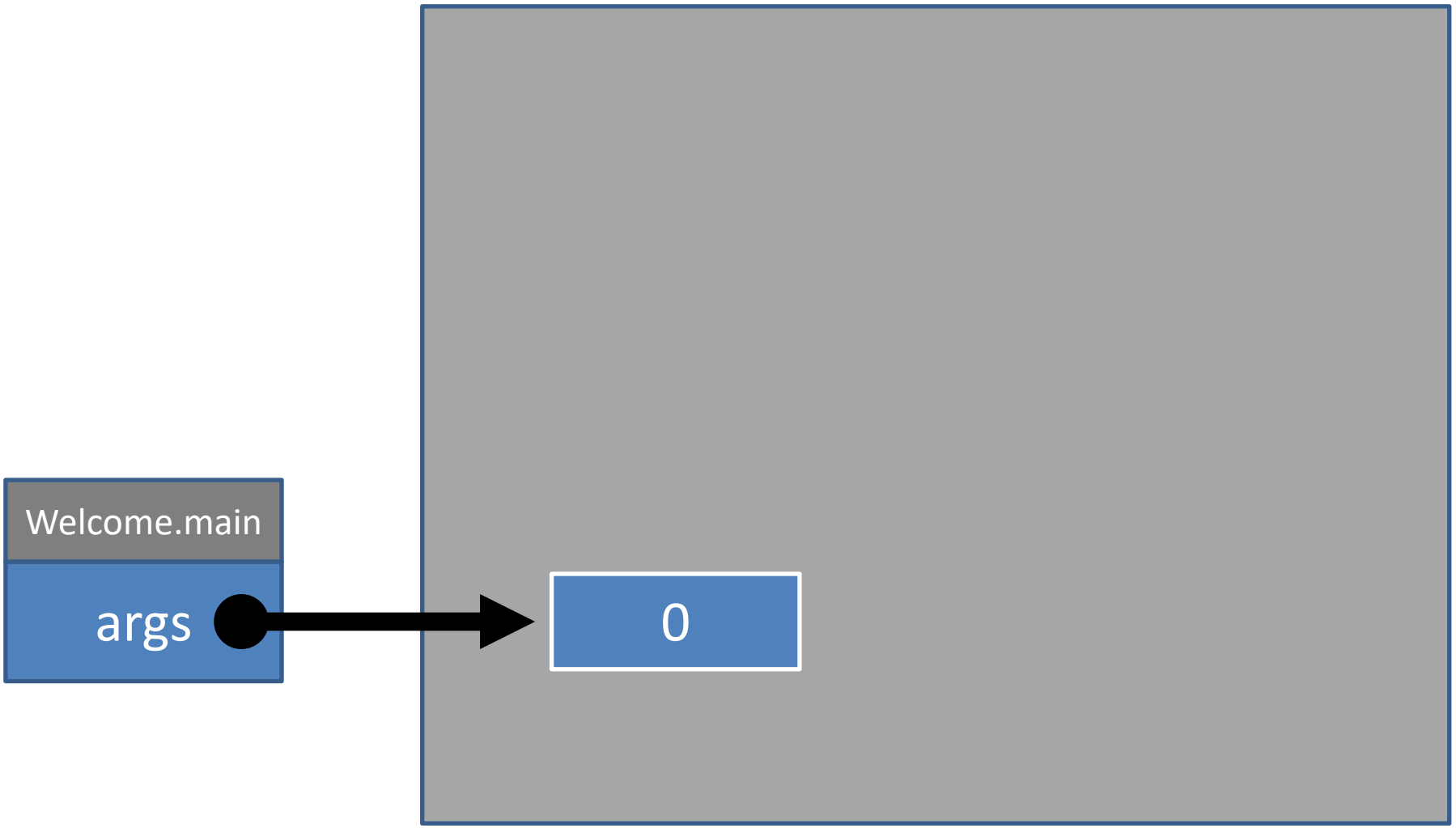

welcome – Welcome.java
Welcome
Git:
Project
Welcome.java
welcome [COS 226]
Welcome
README.md
External Libraries
public class Welcome {
public static void main(String[] args) {
System.out.format("Welcome to CS2!\n");
}
}
Git
TODO
SpotBugs
Terminal
CheckStyle
Event Log
The Checkstyle rules file could not be read (a minute ago)
8:1
LF
UTF-8
2 spaces
main

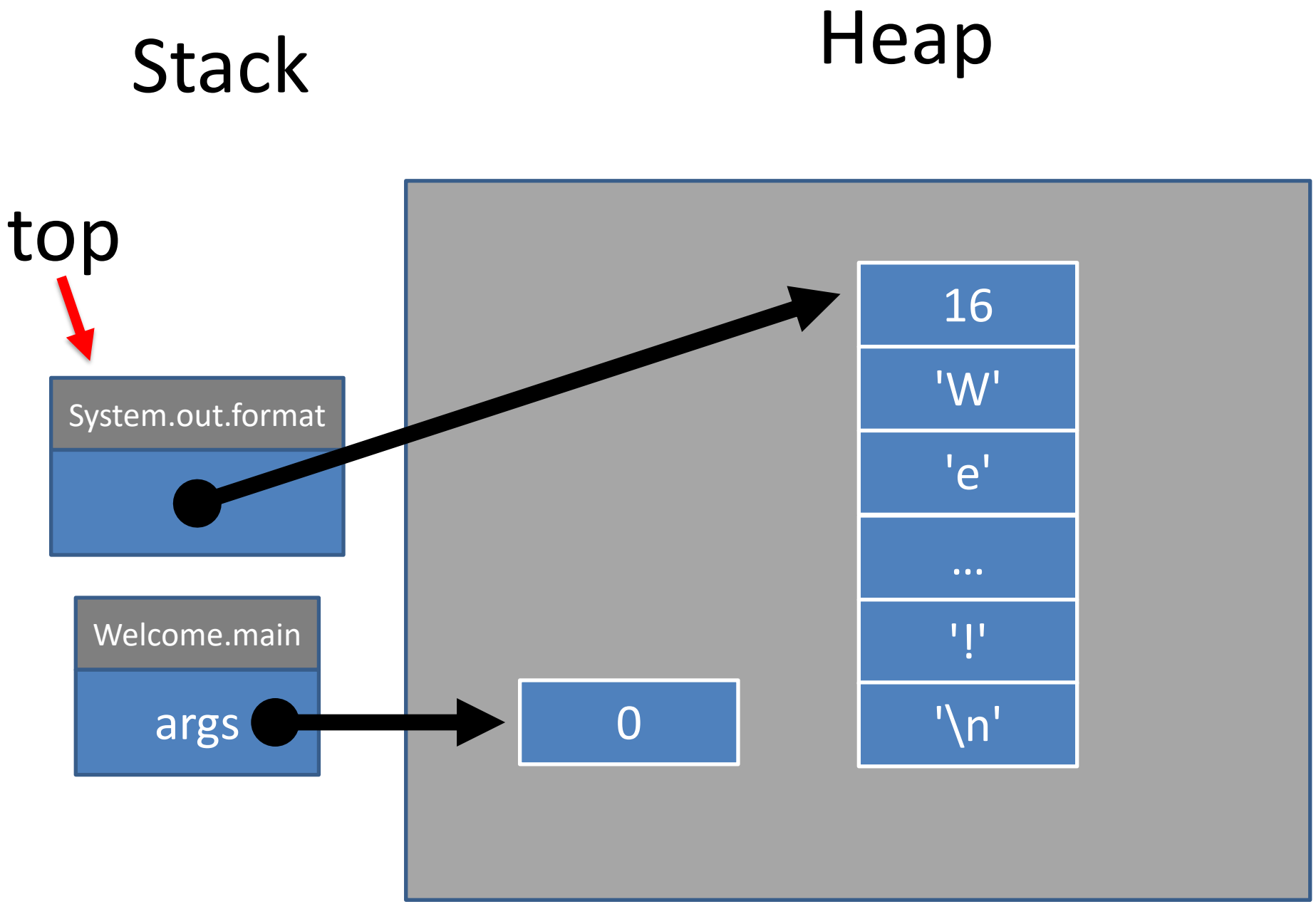
Stack

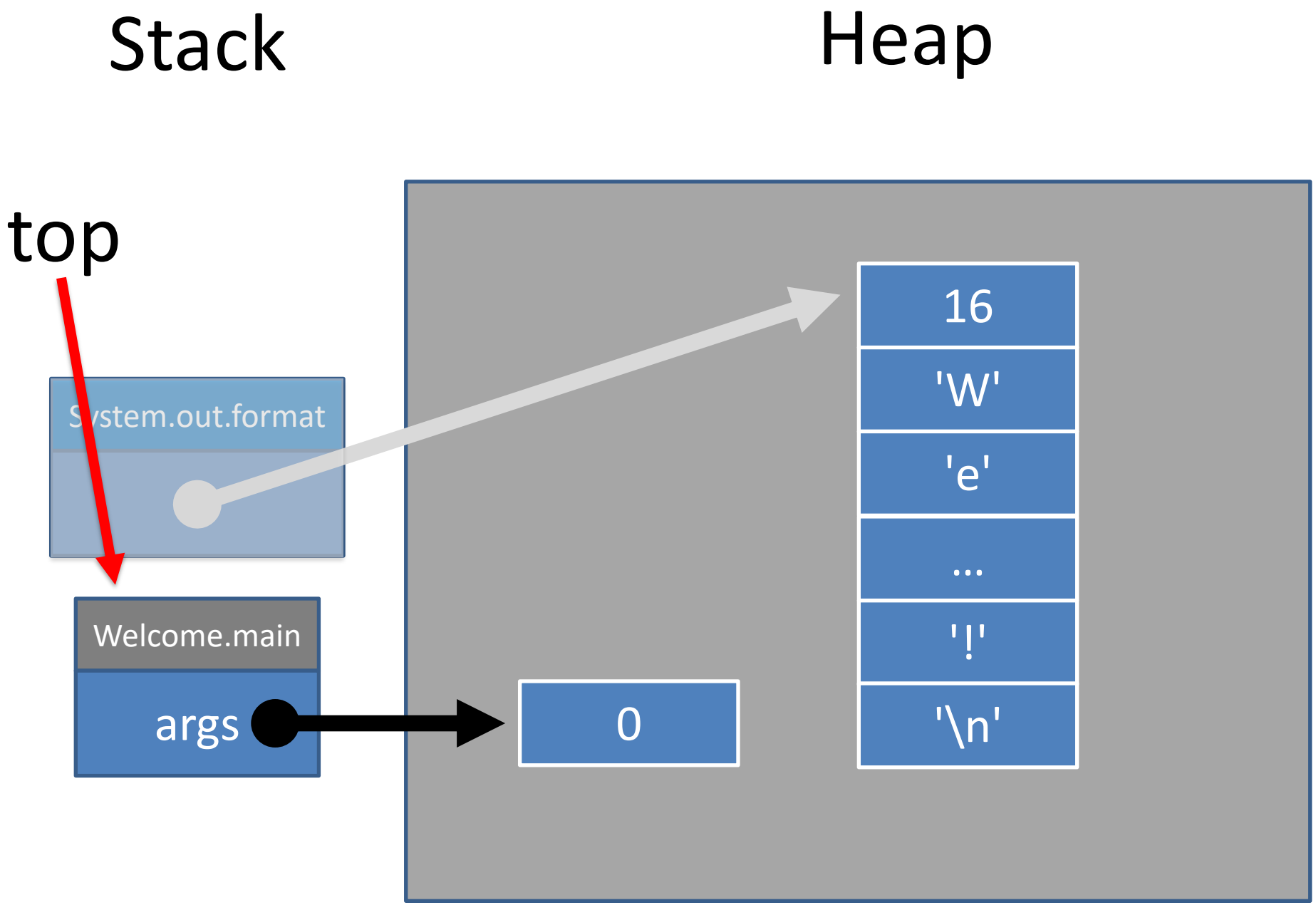
Heap

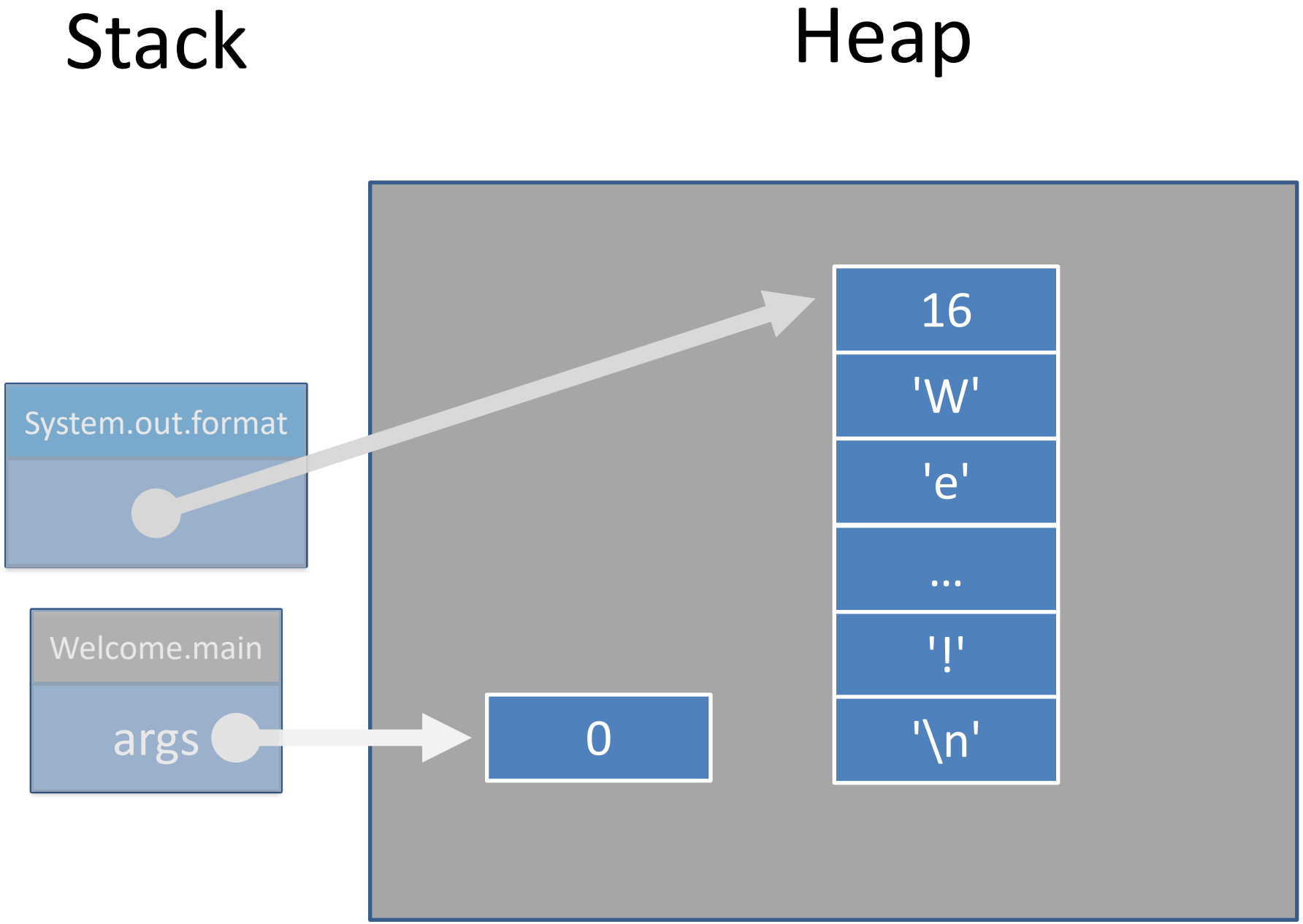
Stack

Heap



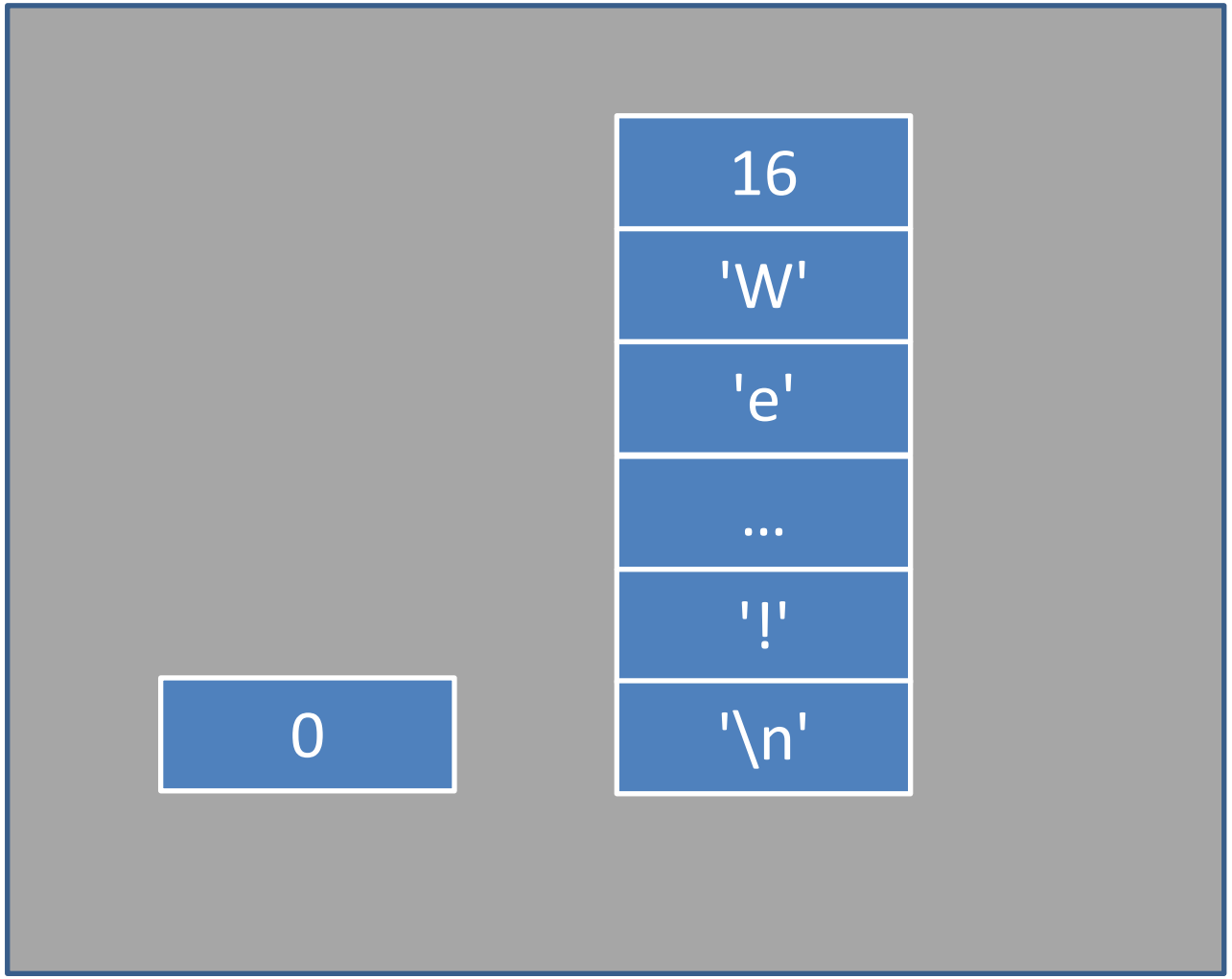

Stack
Heap
top
System.out.format
Welcome.main
args
0
16
'W'
'e'
...
'!'
'\n'


Stack
Heap
top
System.out.format
Welcome.main
args
0
16
'W'
'e'
...
'!'
'\n'


Stack
Heap
System.out.format
Welcome.main
args
0
16
'W'
'e'
...
'!'
'\n'

Stack

Heap



Stack

Heap