Programming Paradigms
Lecture 4: Names, Scopes, and Bindings

Prof. Dr. Michael Pradel
Software Lab, University of Stuttgart
Winter 2019/2020
Names in PLs

Abstraction in two dimensions

- **From hardware**
  - Variable names abstract away how exactly values are stored

- **From implemented functionality**
  - Function names abstract from the implemented behavior
Binding

- Association between entities and their names, e.g.,
  - A variable bound to a memory object
  - A function bound to the code implementing the function

- Different languages have different rules
  - E.g., static vs. dynamic binding
Scope

- **Scope of a binding**: Textual region where binding is active
- **Scope**: Maximal region where no bindings change

Example (Python):

```python
x = 1
def f():
    x = 2
    y = x
```
Scope

- **Scope of a binding**: Textual region where binding is active
- **Scope**: Maximal region where no bindings change

Example (Python):

```python
x = 1
def f():
    x = 2
    y = x
```

- Outer scope
- Scope of function
Overview

- Object lifetime and storage management
- Scopes
- Aliasing and overloading
- Binding of referencing environments
Object Lifetime

Every memory object has a lifetime

- Global variables: Entire program execution
- Local variables: Function execution

Object lifetime vs. binding lifetime

- A single object may be bound to multiple names
- Bindings may be concurrent
Example 1

---

Program execution

---

Value 3

---

\( x \rightarrow y \)

---

fun \( f(x) \)  
\[
\begin{align*}
&x = 3 \\
&\text{return } x \\
&3 \\
y = f(1)
\end{align*}
\]
Example 2:

program execution

object

binding

usually a bug ("dangling reference")

4 use-after-free attack in C
Three kinds of memory objects

- **Static**
  - Absolute address retained throughout execution

- **Stack**
  - Usually within subroutines
  - Allocation/deallocation on call/return

- **Heap**
  - Allocation and deallocation at arbitrary times
Statically Allocated Memory

Depending on the PL, used, e.g., for:

- Global variables
- Constant literals
- Symbol tables
- Program code itself
- Compile-time constants

- Even if local to function
Stack-based Allocation

Arguments to called fcts.

Temporary variables

Misc. for bookkeeping

Return address

fun c() {
    ...
}

fun b(...) {
    if ...
        if ...
            c()
}

fun a c() {
    b c()
}

// main
a c()
Heap-based Allocation

- For dynamically allocated data structures and objects whose size is statically unknown
  - E.g., objects in Java

- Some PLs: Managed memory
  - Unreachable objects: Implicitly deallocated
    - Unreachable = No active binding
  - Less control but fewer bugs
    - E.g., no use-after-free
Quiz: Memory Allocation

Where are the following data objects stored (Java)?

- The integer 23
- The string "John"
- The `Person` object
- The reference variable `p`

```java
class Person {
    int pid;
    String name;
    // constructor
}

public class Driver {
    public static void main(String[] args) {
        int id = 23;
        String pName = "John";
        Person p = null;
        p = new Person(id, pName);
    }
}
```

https://ilias3.uni-stuttgart.de/vote/0ZT9
Quiz: Memory Allocation

class Person {
   int pid;
   String name;

   // constructor
}

class Driver {
   public static void main(String[] args) {
      int id = 23;
      String pName = "John";
      Person p = null;
      p = new Person(id, pName);
   }
}

Stack (in allocation frame of main)
Quiz: Memory Allocation

class Person {
    int pid;
    String name;

    // constructor
}

class Driver {
    public static void main(String[] args) {
        int id = 23;
        String pName = "John";
        Person p = null;
        p = new Person(id, pName);
    }
}