HARVARD
John A. Paulson
School of Engineering
and Applied Sciences

CS153: Compilers
Lecture 17: Compiling Objects

Stephen Chong

https://www.seas.harvard.edu/courses/cs153

Contains content from lecture notes by Steve Zdancewic and Greg Morrisett

Announcements

- HW4: Oat v.1
  - Due Today
- HW5: Oat v.2
  - Released today!
  - Due in 3 weeks
  - Files have much of solution to HW4
  - HW4 last late day is Friday
  - So the files will be released on Canvas Saturday 12am
  - **If you have submitted HW4 and want HW5 files now, email cs153-staff@seas.harvard.edu**
    - We will email you a link to the files

Today

- Overview of HW5
- Object Oriented programming
  - What is it
  - Dynamic dispatch

What Is Object-Oriented Programming?

- Programming based on concept of **objects**, which are **data plus code**
- OOP can be an effective approach to writing large systems
  - Objects naturally model entities
  - OO languages typically support
    - **information hiding** (aka **encapsulation**) to support modularity
    - **inheritance** to support code reuse
- Several families of OO languages:
  - Prototype-based (e.g. Javascript, Lua)
  - Class-based (e.g. C++, Java, C#)
- We focus on the compilation of class-based OO languages

Brief Incomplete History of OO

- (Early 60's) Key concepts emerge in various languages/ programs: sketchpad (Sutherland), SIMSCRIPT (Hoare), and probably many others.
- (1967) Simula 67 (Dahl, Nygaard) crystalizes many ideas (class, object, subclass, dispatch) into a coherent OO language
- (1972) Smalltalk (Kay) introduces the concept of object-oriented programming
- (1978) Modula-2 (Wirth)
- (1985) Eiffel (Meyer)
- (1990's) OO programming becomes mainstream: C++, Java, C#, ...

Classes

- What's the difference between a class and an object?
- A class is a blueprint for objects
- Class typically contains
  - Declared fields / instance variables
    - Values may differ from object to object
    - Usually mutable
  - Methods
    - Shared by all objects of a class
    - Inherited from superclasses
    - Usually immutable
- Methods can be overridden, fields (typically) can not

Example Java Code

```
class Vehicle extends Object {
    int position = 0;
    void move(int x) { this.position += x; }
}
```

```
class Car extends Vehicle {
    int passengers = 0;
    void await(Vehicle v) {
        if (v.position < this.position) {
            v.move(this.position - v.position);
        } else { this.move(10); }
    }
}
```

```
class Truck extends Vehicle {
    void move(int x) { if (x < 55) this.position += x;}
}
```

- Every `Vehicle` is an `Object`
- Every `Car` is a `Vehicle`, every `Truck` is a `Vehicle`
- Every `Vehicle` (and thus every `Car` and `Truck`) have a `position` field and a `move` method
- Every `Car` also has a `passengers` field and an `await` method

Example Java Code

```
class Vehicle extends Object {
    int position = 0;
    void move(int x) { this.position += x; }
}
```

```
class Car extends Vehicle {
    int passengers = 0;
    void await(Vehicle v) {
        if (v.position < this.position) {
            v.move(this.position - v.position);
        } else { this.move(10); }
    }
}
```

```
class Truck extends Vehicle {
    void move(int x) { if (x < 55) this.position += x;}
}
```

- A `Car` can be used anywhere a `Vehicle` is expected (because a `Car` is a `Vehicle`!)
- Class `Truck` **overrides** the `move` method of `Vehicle`
  - Invoking method `o.move(i)` will invoke `Truck`'s `move` method if `o`'s class at run time is `Truck`

Code Generation for Objects

- Methods
  - How do we generate method body code?
  - How do we invoke methods (dispatching)
    - Challenge: handling inheritance
- Fields
  - Memory layout
  - Alignment
  - Challenge: handling inheritance

Need for Dynamic Dispatch

- Methods look like functions. Can they be treated the same?
- Consider the following Java code: Same interface implemented by multiple classes

```
interface IntSet {
    public IntSet insert(int i);
    public boolean has(int i);
    public int size();
}
```

```
class IntSet1 implements IntSet {
  private List<Integer> rep;
  public IntSet1() {
    rep = new LinkedList<Integer>();}

  public IntSet1 insert(int i) {
     rep.add(new Integer(i));
    return this;}

  public boolean has(int i) {
    return rep.contains(new Integer(i));}

  public int size() {return rep.size();}
}
```

```
class IntSet2 implements IntSet {
  private Tree rep;
  private int size;
  public IntSet2() {
    rep = new Leaf(); size = 0;}

  public IntSet2 insert(int i) {
     Tree nrep = rep.insert(i);
    if (nrep != rep) {
      rep = nrep; size += 1;
    }
     return this;}

  public boolean has(int i) {
     return rep.find(i);}

  public int size() {return size;}
}
```

Need for Dynamic Dispatch

```
interface IntSet {
    public IntSet insert(int i);
    public boolean has(int i);
    public int size();
}
```

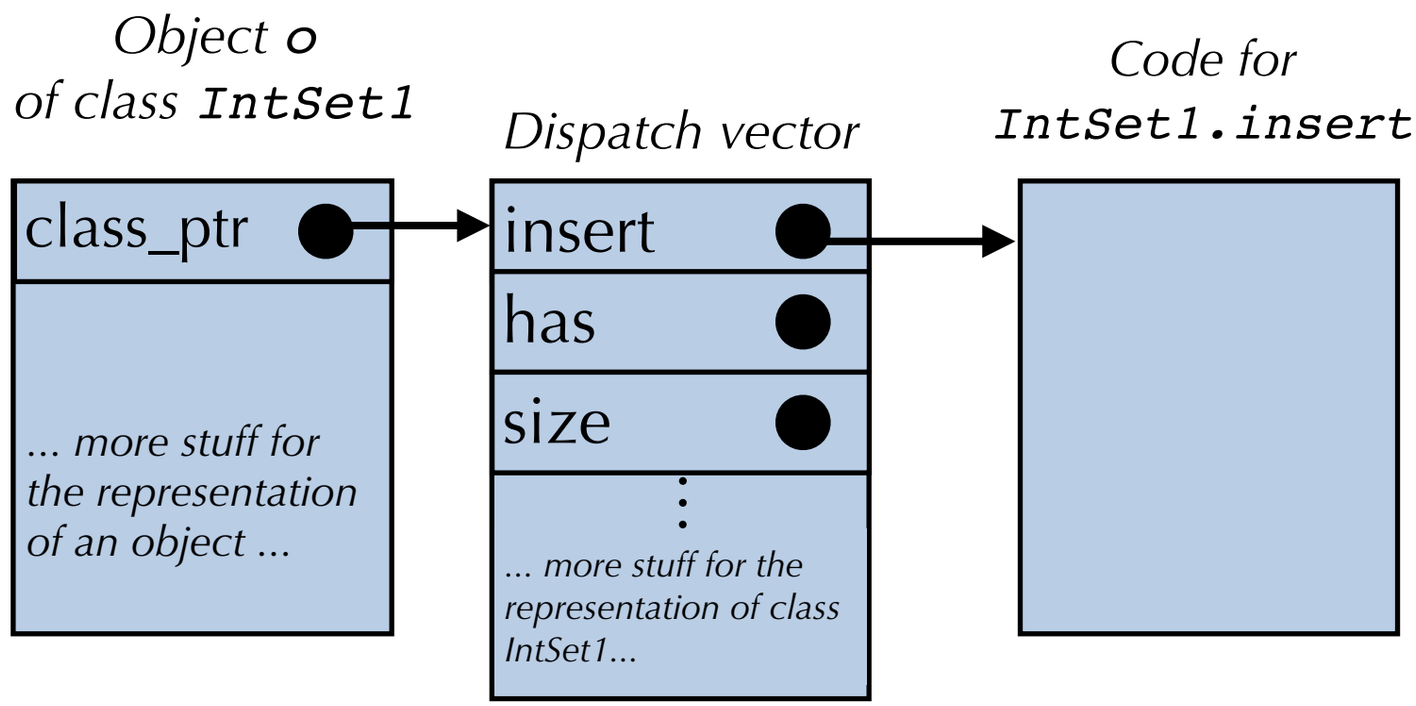
- Suppose a client uses the IntSet interface

```
IntSet set = foo();
int x = set.size();
```

- Which code to call?
  - `IntSet1.size`? `IntSet2.size`?
- Client code doesn't know which code! Could be either at runtime.
  - Objects must "know" which code to call
  - Invocation of method must indirect through object

Dynamic Dispatch Solution

- So we need some way at run time to figure out which code to invoke
- Solution: **dispatch table** (aka **virtual method table**, **vtable**)
  - Each class has table (array) of function pointers
  - Each method of class is at a known index of table




```
IntSet set = foo();
int x = set.size();
```