Where are we?

5. Storage

5.4. Value vs. reference semantics
5. Storage
5.4. Value vs. reference semantics

Variables: reference vs. value semantics

Value Semantics.

- Variable contains the actual value.
- C, C++
- JAVA for builtin, atomic types

Reference Semantics.

- Variable contains a reference to a value which is stored elsewhere.
- C, C++, if pointers or references are used
- JAVA for all other types, including arrays
- Most modern languages
Values vs. reference semantics in JAVA

The basic type system of JAVA is defined by:

- **8 atomic types**: byte, short, int, long, float, double, boolean, char
- **1 pseudo type**: void

4 type constructors:

- array
- class
- interface
- enum

- Precisely 8 types in JAVA follow value semantics
- All the rest are reference semantics
5. Storage

5.4. Value vs. reference semantics

Wrapper classes

- When generics were introduced to Java, it was discovered that the implementation was much simpler for reference types.
- The 8 value types did not justify extra machinery:
- Instead, the Java library introduced reference type equivalents

  Integral types  Byte, Short, Integer, Long
  Floating point types  Float, Double
  Other types  Boolean, Character
  Unit types  Void

```
List<double> ds1; // compilation error; double is not a reference type
List<Double> ds2; // Double is a reference type
```
“Integer” vs. “int” in JAVA

- Each wrapper class (except for Void) wraps a value of the corresponding primitive type.
- Wrapper types are almost fully interchangeable with their primitive equivalents:

```java
int v = 3; // Primitive type
Integer r = new Integer(a); // Wrapper type
v = r.intValue(); // Explicit conversion
v = r; // auto un-boxing
r = v; // auto boxing
```

Auto boxing  Coercion from, e.g., int to Integer
Auto unboxing Coercion from, e.g., Integer to int

- Type Integer also includes value null
- Type int does not include value null
- The following will generate RuntimeException

```java
Double dd = null;
double d = d;
```
The OO terminology

Objects?

- *OO languages often use the term “object”...*
The OO terminology

Objects?

- *OO languages often use the term “object”...*
- *the term propagates also to non-OO PLs*

Object’s `i.d.`,

```
Variable/Object
foo      bar      baz
```

```
Variable/Value
3.1415926
```
The OO terminology

Objects?

- **OO languages often use the term “object”...**
- *the term propagates also to non-OO PLs*
- *means (usually) a “variable” whose contents has an i.d.*

Object’s *i.d.*,}

```plaintext
Variable/Object

foo  bar  baz

Variable/Value

3.1415926
```
The OO terminology

Objects?

- *OO languages often use the term “object”...*
- *the term propagates also to non-OO PLs*
- *means (usually) a “variable” whose contents has an i.d.*
- *total inspection of this object, yields a value with an i.d.,*

Object’s *i.d.*,
The OO terminology

Objects?

- **OO languages often use the term “object”...**
- *the term propagates also to non-OO PLs*
- *means (usually) a “variable” whose contents has an i.d.*
- *total inspection of this object, yields a value with an i.d.,*

```
Variable
  a
```

```
Variable/Object
  foo
  bar
  baz
```

```
Variable/Value
  3.1415926
```

Object’s i.d.,
The OO terminology

Objects?

- *OO languages often use the term “object”...*
- *the term propagates also to non-OO PLs*
- *means (usually) a “variable” whose contents has an i.d.*
- *total inspection of this object, yields a value with an i.d.*, 

Object’s *i.d.*,

- cannot be changed by user
The OO terminology

Objects?

- **OO languages often use the term “object”…**
- **the term propagates also to non-OO PLs**
- **means (usually) a “variable” whose contents has an i.d.**
- **total inspection of this object, yields a value with an i.d.,**

Object’s *i.d.*,

- cannot be changed by user
- two object with the same contents, still have a distinct *i.d.*
C: value semantic of assignment

A “date” record in C

```c
typedef struct Date {
    int year, month, day;
} Date;
```

Initializing two variables of this type:
```c
Date today = {2015, 01, 07};
Date tomorrow = {2015, 01, 08};
```
C: value semantic of assignment

A “date” record in C

```c
typedef struct Date {
    int year, month, day;
} Date;
```

Initializing two variables of this type:
```c
Date today = {2015, 01, 07};
Date tomorrow = {2015, 01, 08};
```

today = tomorrow;
C: value semantic of assignment

A “date” record in C

```c
typedef struct Date {
  int year, month, day;
} Date;
```

Initializing two variables of this type:

```c
Date today = {2015,01,07};
Date tomorrow = {2015,01,08};
```

today = tomorrow;

Before Assignment

<table>
<thead>
<tr>
<th>today</th>
<th>tomorrow</th>
</tr>
</thead>
<tbody>
<tr>
<td>2015 01 07</td>
<td>2015 01 08</td>
</tr>
</tbody>
</table>

year month day  year month day
C: value semantic of assignment

Initializing two variables of this type:
Date today = {2015,01,07};
Date tomorrow = {2015,01,08};

today = tomorrow;

After Assignment

Before Assignment

<table>
<thead>
<tr>
<th></th>
<th>today</th>
<th>tomorrow</th>
</tr>
</thead>
<tbody>
<tr>
<td>year</td>
<td>2015</td>
<td>2015</td>
</tr>
<tr>
<td>month</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>day</td>
<td>07</td>
<td>08</td>
</tr>
</tbody>
</table>

A “date” record in C

typedef struct Date {
    int year, month, day;
} Date;
C: value semantic of assignment

```c
typedef struct Date {
    int year, month, day;
} Date;

Date today = {2015, 01, 07};
Date tomorrow = {2015, 01, 08};

today = tomorrow;
```

Before Assignment

<table>
<thead>
<tr>
<th>today</th>
<th>tomorrow</th>
</tr>
</thead>
<tbody>
<tr>
<td>2015</td>
<td>2015</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>07</td>
<td>08</td>
</tr>
<tr>
<td>yearmonthday</td>
<td>yearmonthday</td>
</tr>
</tbody>
</table>

After Assignment

<table>
<thead>
<tr>
<th>today</th>
<th>tomorrow</th>
</tr>
</thead>
<tbody>
<tr>
<td>2015</td>
<td>2015</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>08</td>
<td>08</td>
</tr>
<tr>
<td>yearmonthday</td>
<td>yearmonthday</td>
</tr>
</tbody>
</table>
Reference semantic of assignment in JAVA

A “date” record in JAVA

```java
class Date {
    Date(int year, int month, int day) {
        this.year = year;
        this.month = month;
        this.day = day;
    }
    int year, month, day;
}

Date today =
    new Date(2015, 01, 07);

Date tomorrow =
    new Date(2015, 01, 08);
```

Before Assignment

After Assignment
Reference semantic of assignment in JAVA

A “date” record in JAVA

class Date {
    Date(int year, int month, int day) {
        this.year = year;
        this.month = month;
        this.day = day;
    }
    int year, month, day;
}

Date today =
    new Date(2015, 01, 07);

Date tomorrow =
    new Date(2015, 01, 08);

today = tomorrow;

Before Assignment

After Assignment
Reference semantic of assignment in JAVA

A “date” record in JAVA

```java
class Date {
    Date(int year, int month, int day) {
        this.year = year;
        this.month = month;
        this.day = day;
    }
    int year, month, day;
}
```

```java
Date today =
    new Date(2015, 01, 07);
Date tomorrow =
    new Date(2015, 01, 08);
today = tomorrow;
```

Before Assignment       After Assignment
Reference semantic of assignment in JAVA

A "date" record in JAVA

```java
class Date {
    Date(int year, int month, int day) {
        this.year = year;
        this.month = month;
        this.day = day;
    }
    int year, month, day;
}
```

Date today =
   new Date(2015, 01, 07);

Date tomorrow =
   new Date(2015, 01, 08);

today = tomorrow;

Before Assignment                                      After Assignment
Reference semantic of assignment in JAVA

A "date" record in JAVA

class Date {
    Date(int year, int month, int day) {
        this.year = year;
        this.month = month;
        this.day = day;
    }
    int year, month, day;
}

Date today =
    new Date(2015, 01, 07);

Date tomorrow =
    new Date(2015, 01, 08);

today = tomorrow;

Before Assignment

After Assignment
Reference semantic of assignment in **JAVA**

A “date” record in **JAVA**

```java
class Date {
    Date(int year, int month, int day) {
        this.year = year;
        this.month = month;
        this.day = day;
    }
    int year, month, day;
}
```

```java
Date today =
    new Date(2015, 01, 07);
Date tomorrow =
    new Date(2015, 01, 08);
today = tomorrow;
```

Before Assignment

Before Assignment

<table>
<thead>
<tr>
<th>today</th>
<th>tomorrow</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Date today](2015 01 07)</td>
<td>![Date tomorrow](2015 01 08)</td>
</tr>
</tbody>
</table>

After Assignment

After Assignment

<table>
<thead>
<tr>
<th>today</th>
<th>tomorrow</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Date today](2015 01 07)</td>
<td>![Date tomorrow](2015 01 08)</td>
</tr>
</tbody>
</table>
Reference semantic of assignment in Java

A “date” record in Java

```java
class Date {
    Date(int year, int month, int day) {
        this.year = year;
        this.month = month;
        this.day = day;
    }
    int year, month, day;
}
```

Before Assignment

```
today = tomorrow;
```

After Assignment

```
Date today =
    new Date(2015, 01, 07);
Date tomorrow =
    new Date(2015, 01, 08);
```
C++ vs. JAVA

C++: Value semantics

```cpp
class Date {public:
    int year, month, day;
    Date(int year, int month, int day) {
        this->year = year;
        this->month = month;
        this->day = day;
    }
};

Date today(2015, 01, 07);
Date tomorrow(2015, 01, 08);
today = tomorrow;
tomorrow.year = 3025;
cout << today.year;
```

JAVA: Reference semantics

```java
class Date {
    int year, month, day;
    Date(int year, int month, int day) {
        this.year = year;
        this.month = month;
        this.day = day;
    }
}

Date today = new Date(2015, 01, 07);
Date tomorrow = new Date(2015, 01, 08);
today = tomorrow;
tomorrow.year = 3025;
System.out.print(today.year);
```

1 Can you detect and explain all the syntactical differences between the two languages?
C++ vs. Java

C++: Value semantics

```cpp
class Date {
  public:
    int year, month, day;
    Date(int year, int month, int day) {
      this->year = year;
      this->month = month;
      this->day = day;
    }

  Date today(2015, 01, 07);
  Date tomorrow(2015, 01, 08);

  today = tomorrow;
  tomorrow.year = 3025;
  cout << today.year;
}
```

Java: Reference semantics

```java
class Date {
  int year, month, day;
  Date(int year, int month, int day) {
    this.year = year;
    this.month = month;
    this.day = day;
  }

  Date today = new Date(2015, 01, 07);
  Date tomorrow = new Date(2015, 01, 08);

  today = tomorrow;
  tomorrow.year = 3025;
  System.out.print(today.year);
}
```

1 Can you detect and explain all the syntactical differences between the two languages?
5.4. Value vs. reference semantics

### C++: Value semantics

```cpp
class Date {
public:
    int year, month, day;
    Date(int year, int month, int day) {
        this->year = year;
        this->month = month;
        this->day = day;
    }
};
```

```cpp
date today(2015, 01, 07);
date tomorrow(2015, 01, 08);
today = tomorrow;
tomorrow.year = 3025;
```

```cpp
cout << today.year;
```

```cpp
C++ notation: "->"
```

### Java: Reference semantics

```java
class Date {
    int year, month, day;
    Date(int year, int month, int day) {
        this.year = year;
        this.month = month;
        this.day = day;
    }
};
```

```java
date today = new Date(2015, 01, 07);
date tomorrow = new Date(2015, 01, 08);
today = tomorrow;
tomorrow.year = 3025;
```

```java
System.out.print(today.year);
```

```java
Java notation: "."
```

### C++ Legacy

```cpp
class Date {
    int year, month, day;
    Date(int year, int month, int day) {
        this->year = year;
        this->month = month;
        this->day = day;
    }
};
```

```cpp
date today(2015, 01, 07);
date tomorrow(2015, 01, 08);
today = tomorrow;
tomorrow.year = 3025;
cout << today.year;
```

### Java Legacy

```java
class Date {
    int year, month, day;
    Date(int year, int month, int day) {
        this.year = year;
        this.month = month;
        this.day = day;
    }
};
```

```java
date today = new Date(2015, 01, 07);
date tomorrow = new Date(2015, 01, 08);
today = tomorrow;
tomorrow.year = 3025;
System.out.print(today.year);
```

---

1 Can you detect and explain all the syntactical differences between the two languages?
C++: Value semantics

```cpp
class Date {
public:
  int year, month, day;
  Date(int year, int month, int day) {
    this->year = year;
    this->month = month;
    this->day = day;
  }

  Date today(2015, 01, 07);
  Date tomorrow(2015, 01, 08);

today = tomorrow;
tomorrow.year = 3025;
cout << today.year;
```

Java: Reference semantics

```java
class Date {
  int year, month, day;
  Date(int year, int month, int day) {
    this.year = year;
    this.month = month;
    this.day = day;
  }

  Date today = new Date(2015, 01, 07);
  Date tomorrow = new Date(2015, 01, 08);

today = tomorrow;
tomorrow.year = 3025;
System.out.print(today.year);
```

1. Can you detect and explain all the syntactical differences between the two languages?
C++: Value semantics

```cpp
class Date {public:
    int year, month, day;
    Date(int year, int month, int day) {
        this->year = year;
        this->month = month;
        this->day = day;
    }
};
```

```cpp
Date today(2015, 01, 07);
Date tomorrow(2015, 01, 08);
today = tomorrow;
tomorrow.year = 3025;
cout << today.year;
```

Java: Reference semantics

```java
class Date {
    int year, month, day;
    Date(int year, int month, int day) {
        this.year = year;
        this.month = month;
        this.day = day;
    }
};
```

```java
Date today = new Date(2015, 01, 07);
Date tomorrow = new Date(2015, 01, 08);
today = tomorrow;
tomorrow.year = 3025;
System.out.print(today.year);
```

---

\(^1\) Can you detect and explain all the syntactical differences between the two languages?
Comparing the two semantics

C++: Value Semantic

```cpp
// Creating values for today and tomorrow
Date today = Date(2015, 01, 07);
Date tomorrow = Date(2015, 01, 08);
// Assigning variable tomorrow to today
today = tomorrow;
tomorrow.year = 3025;
cout << today.year;
```

<table>
<thead>
<tr>
<th>today</th>
<th>tomorrow</th>
</tr>
</thead>
<tbody>
<tr>
<td>2015</td>
<td>3025</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>08</td>
<td>08</td>
</tr>
</tbody>
</table>

Java: Reference Semantic

```java
// Creating values for today and tomorrow
Date today = new Date(2015, 01, 07);
Date tomorrow = new Date(2015, 01, 08);
// Assigning variable tomorrow to today
today = tomorrow;
tomorrow.year = 3025;
System.out.print(today.year);
```

Output is...

<table>
<thead>
<tr>
<th>today</th>
<th>tomorrow</th>
</tr>
</thead>
<tbody>
<tr>
<td>2015</td>
<td>3025</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>08</td>
<td>08</td>
</tr>
</tbody>
</table>
Comparing the two semantics

### C++: Value Semantic

```cpp
// Creating values for today and tomorrow
Date today = Date(2015, 01, 07);
Date tomorrow = Date(2015, 01, 08);
// Assigning variable tomorrow to today
today = tomorrow;
tomorrow.year = 3025;
cout << today.year;
```

**today**

<table>
<thead>
<tr>
<th>year</th>
<th>month</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>2015</td>
<td>01</td>
<td>08</td>
</tr>
</tbody>
</table>

**tomorrow**

<table>
<thead>
<tr>
<th>year</th>
<th>month</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>3025</td>
<td>01</td>
<td>08</td>
</tr>
</tbody>
</table>

**Output is...**

2015

### Java: Reference Semantic

```java
// Creating values for today and tomorrow
Date today = new Date(2015, 01, 07);
Date tomorrow = new Date(2015, 01, 08);
// Assigning variable tomorrow to today
today = tomorrow;
tomorrow.year = 3025;
System.out.print(today.year);
```

**today**

<table>
<thead>
<tr>
<th>year</th>
<th>month</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>2015</td>
<td>01</td>
<td>01</td>
</tr>
</tbody>
</table>

**tomorrow**

<table>
<thead>
<tr>
<th>year</th>
<th>month</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>3025</td>
<td>01</td>
<td>08</td>
</tr>
</tbody>
</table>
Comparing the two semantics

### C++: Value Semantic

```cpp
// Creating values for today and tomorrow
Date today = Date(2015, 01, 07);
Date tomorrow = Date(2015, 01, 08);
// Assigning variable tomorrow to today
today = tomorrow;
tomorrow.year = 3025;
cout << today.year;
```

**Output is...** 2015

<table>
<thead>
<tr>
<th>today</th>
<th>tomorrow</th>
</tr>
</thead>
<tbody>
<tr>
<td>2015</td>
<td>3025</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>08</td>
<td>08</td>
</tr>
</tbody>
</table>

### JAVA: Reference Semantic

```java
// Creating values for today and tomorrow
Date today = new Date(2015, 01, 07);
Date tomorrow = new Date(2015, 01, 08);
// Assigning variable tomorrow to today
today = tomorrow;
tomorrow.year = 3025;
System.out.print(today.year);
```

**Output is...** 3025

<table>
<thead>
<tr>
<th>today</th>
<th>tomorrow</th>
</tr>
</thead>
<tbody>
<tr>
<td>2015</td>
<td>3025</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>01</td>
<td>08</td>
</tr>
</tbody>
</table>

J. Gil (Technion–IIT)
Comparing the two semantics

### C++: Value Semantic

```cpp
// Creating values for today and tomorrow
Date today = Date(2015, 01, 07);
Date tomorrow = Date(2015, 01, 08);
// Assigning variable tomorrow to today
today = tomorrow;
tomorrow.year = 3025;
cout << today.year;
```

**Output is...**

```
2015
```

### JAVA: Reference Semantic

```java
// Creating values for today and tomorrow
Date today = new Date(2015, 01, 07);
Date tomorrow = new Date(2015, 01, 08);
// Assigning variable tomorrow to today
today = tomorrow;
tomorrow.year = 3025;
System.out.print(today.year);
```

**Output is...**

```
3025
```
Comparing the two semantics

**C++: Value Semantic**

```c++
// Creating values for today and tomorrow
Date today = Date(2015, 01, 07);
Date tomorrow = Date(2015, 01, 08);
// Assigning variable tomorrow to today
today = tomorrow;
tomorrow.year = 3025;
cout << today.year;
```

**JAVA: Reference Semantic**

```java
// Creating values for today and tomorrow
Date today = new Date(2015, 01, 07);
Date tomorrow = new Date(2015, 01, 08);
// Assigning variable tomorrow to today
today = tomorrow;
tomorrow.year = 3025;
System.out.print(today.year);
```

**Output is...**

C++: 2015

JAVA: 3025
Where are we?

5. Storage

5.4. Value vs. reference semantics

5.4.1. Shared representation & lazy copy
5. Storage

5.4. Value vs. reference semantics

5.4.1. Shared representation & lazy copy
Which semantic does ML use?

**Answer**

The programmer shouldn’t care and cannot know!

- *It looks like value semantics.* In reality, ML, **Lisp** and many other languages use *value semantics*, in the sense that the programmer cannot observe any “references” in the program.

- *Implementation is with references.* Behind the scenes, memory and time is saved by using references.
Efficient **LISP** implementation with references

Let \( \alpha \), \( \beta \) be two large S-expressions. 

\[
\begin{align*}
\text{(setq } a \alpha \text{)} & \\
\text{(setq } b \beta \text{)} & \\
\text{(cons } a b \text{)} & \\
\text{(setq } c \text{ (cons } a b \text{)} &
\end{align*}
\]

\( \alpha \) Some large S-expression

\( \beta \) Some other large S-expression

\( a \) \( b \) \( \text{cons} \)
Efficient **LISP** implementation with references

- Let $\alpha, \beta$ be two large $S$-expressions.

```
(let ((a \alpha))
  (let ((b \beta))
    (cons a b)
    (setq c (cons a b))
)
```
Efficient **LISP** implementation with references

- Let $\alpha, \beta$ be two large $S$-expressions.
- (setq a $\alpha$)
Efficient LISP implementation with references

- Let $\alpha, \beta$ be two large S-expressions.
- `(setq a \alpha)`
- `(setq b \beta)`

\[ \alpha \] Some large S-expression
\[ \beta \] Some other large S-expression
Efficient **LISP** implementation with references

- Let \( \alpha, \beta \) be two large S-expressions.
- \((\text{setq } a \ \alpha)\)
- \((\text{setq } b \ \beta)\)
- \((\text{cons } a \ b)\)
Efficient **LISP** implementation with references

- Let $\alpha, \beta$ be two large S-expressions.
- `(setq a \alpha)`
- `(setq b \beta)`
- `(cons a b)`
- `(setq c (cons a b))`
Generalization

**Definition (Lazy Copying)**

*Generalizing the Lisp approach*, lazy copying is an implementation technique of value semantics, where a copy of a large object is made by creating a new reference to it. The actual copy operation is made when (and if) the source or the destination variables are modified.

- Generalizes the Lisp approach
- Support for languages which permits mutation of values.
- Many extensions of pure-Lisp allow such mutation
- Mutation is the bread an butter of imperative programming.
- Conceptually similar to “copy on write” in memory management
Reference semantic in C/C++?

Of course!
You just have to be explicit about it!

class Date {...};

Date *today = new Date(2015, 01, 07);
Date *tomorrow = new Date(2015, 01, 08);
today = tomorrow; // Leak!
tomorrow->year = 3025;
cout << today->year;
delete tomorrow;
delete today; // Heap corruption?

J. Gil (Technion–IIT)
CS 234319: Programming Languages
January 7, 2015  Section 5.4: p. 13/26
Reference semantic in C/C++?

Of course!

You just have to be explicit about it!

class Date {...};

Date *today = new Date(2015, 01, 07);
Date *tomorrow = new Date(2015, 01, 08);
today = tomorrow;    // Leak!
tomorrow->year = 3025;
cout << today->year;
delete tomorrow;
delete today;       // Heap corruption?

Output is...
Reference semantic in C/C++?

Of course!
You just have to be explicit about it!

class Date {...};

Date *today = new Date(2015, 01, 07);
Date *tomorrow = new Date(2015, 01, 08);
today = tomorrow; // Leak!
tomorrow->year = 3025;
cout << today->year;
delete tomorrow;
delete today; // Heap corruption?

Output is... 3025
5. Storage

5.4. Value vs. reference semantics / 5.4.1. Shared representation & lazy copy

Value semantic in JAVA?

Of course!
You just have to be explicit about it!

class Date implements Cloneable {
    :
    }

Date today = new Date(2015, 01, 07);
Date tomorrow = new Date(2015, 01, 08);

today = (Date) tomorrow.clone();
tomorrow.year = 3025;
System.out.println(today.year);
Value semantic in Java?

Of course!
You just have to be explicit about it!

class Date implements Cloneable {
    :
    
}

Date today = new Date(2015, 01, 07);
Date tomorrow = new Date(2015, 01, 08)

    today = (Date) tomorrow.clone();
    tomorrow.year = 3025;
    System.out.println(today.year);

Output is...
Value semantic in Java?

Of course!
You just have to be explicit about it!

class Date implements Cloneable {
    :
    }

Date today = new Date(2015, 01, 07);
Date tomorrow = new Date(2015, 01, 08);

today = (Date) tomorrow.clone();
tomorrow.year = 3025;
System.out.println(today.year);

Output is... 2015
Assignment in reference semantic languages?

But, ... what does

(Date) tomorrow.clone();

actually mean?

More generally, in any reference semantic programming language:
Assignment in reference semantic languages?

But,... what does

(Date) tomorrow.clone();

actually mean?

More generally, in any reference semantic programming language:

- Given two variables, \(a\) and \(b\),

\[ a \quad b \]
Assignment in reference semantic languages?

But,... what does

(Date) tomorrow.clone();

actually mean?

More generally, in any reference semantic programming language:

- Given two variables, a and b,
- each containing a reference to a value,
Assignment in reference semantic languages?

But,... what does

(Date) tomorrow.clone();

actually mean?

More generally, in any reference semantic programming language:

- Given two variables, a and b,
- each containing a reference to a value,
- which may include references to a network of variables,
Assignment in reference semantic languages?

But,... what does

(Date) tomorrow.clone();

actually mean?

More generally, in any reference semantic programming language:

- Given two variables, \( a \) and \( b \),
- each containing a reference to a value,
- which may include references to a network of variables,
- and an assignment command
  \[
  a := b;
  \]
Assignment in reference semantic languages?

But, ... what does

(Date) tomorrow.clone();

actually mean?

More generally, in any reference semantic programming language:

- Given two variables, \( a \) and \( b \),
- each containing a reference to a value,
- which may include references to a network of variables,
- and an assignment command
  \[
  a := b;
  \]
- what's going to happen?
Assignment in reference semantic languages?

2 Variables...
Assignment in reference semantic languages?

1 Assignment...
Assignment in reference semantic languages?

3 Semantics...
Assignment in reference semantic languages?

Reference Assignment

1st Alternative
Assignment in reference semantic languages?

Reference Assignment

*Only the reference is copied*

1st Alternative
Assignment in reference semantic languages?

Reference Assignment

*Only the reference is copied*

1st Alternative
Assignment in reference semantic languages?

Other alternatives?
Assignment in reference semantic languages?

Deep Clone

2nd Alternative
Assignment in reference semantic languages?

Deep Clone

The whole network of variables accessible from \( b \) is duplicated, and assigned to \( a \)

2nd Alternative
Assignment in reference semantic languages?

Deep Clone

The whole network of variables accessible from \( b \) is duplicated, and assigned to \( a \)

2nd Alternative
Assignment in reference semantic languages?

Is there a midway?
Assignment in reference semantic languages?

Shallow Copy

3rd Alternative
Assignment in reference semantic languages?

**Shallow Copy**

*Only the variable is copied*

**3rd Alternative**
Assignment in reference semantic languages?

**3rd Alternative**

**Shallow Copy**

*Only the variable is copied*
Assignment in reference semantic languages?

Let's repeat
Assignment in reference semantic languages?

Reference Assignment

Deep Clone

Shallow Copy

3 Semantics
Assignment in reference semantic languages?

Reference Assignment

*Only the reference is copied*

Deep Clone

*The whole network of variables accessible from b is duplicated, and assigned to a*

Shallow Copy

*Only the variable is copied*

3 Rules
Assignment in reference semantic languages?

Reference Assignment

*Only the reference is copied*

Deep Clone

*The whole network of variables accessible from b is duplicated, and assigned to a*

Shallow Copy

*Only the variable is copied*

3 Different Outcomes
Assignment in reference semantic languages?

**Reference Assignment**

*Only the reference is copied*

![Diagram of Reference Assignment]

**Deep Clone**

*The whole network of variables accessible from b is duplicated, and assigned to a*

![Diagram of Deep Clone]

**Shallow Copy**

*Only the variable is copied*

![Diagram of Shallow Copy]

More...
Yet another semantic... shallow clone

The variable itself is cloned, but all the references in it are copied, rather than being cloned

Before Assignment

After Assignment
Summary: assignment/copy semantics

Reference Semantic

Shallow Copy

Shallow Clone

Deep Clone
### Comparison of assignment strategies

<table>
<thead>
<tr>
<th>Semantic</th>
<th>Null pointer assignment?</th>
<th>Memory Allocation?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference Assignment</td>
<td>Never</td>
<td>✗</td>
</tr>
<tr>
<td>Shallow Copy</td>
<td>Maybe</td>
<td>✗</td>
</tr>
<tr>
<td>Shallow Clone</td>
<td>Never</td>
<td>✓ (bounded)</td>
</tr>
<tr>
<td>Deep Clone</td>
<td>Never</td>
<td>✓ (unbounded)</td>
</tr>
</tbody>
</table>
Where are we?

5. Storage

5.4. Value vs. reference semantics

5.4.2. Value vs. reference semantics in various PLs
Where are we?

5. Storage

5.4. Value vs. reference semantics

5.4.2. Value vs. reference semantics in various PLs
What does Java clone() do?

**Runtime Exception**
- if the class *does not implement* interface Cloneable

**Shallow Clone**
- if the class implements interface Cloneable, and
- the programmer *does not override* the default clone() method.

**Whatever**
- if the class implements interface Cloneable, and
- the programmer overrides the default clone method
- *in whatever way he likes.*

**Deep Clone**
- if the class implements interface Cloneable, and
- the programmer overrides the default clone method, and
- *correctly implements a “Deep Clone” semantic*
Working knowledge of semantics?

Typical Exam Question: Read the documentation of a particular language feature, and determine which semantic it uses.

Feature could be “assignment” (of a particular kind of variable), “library function”, and even “equality testing”: comparison of components one by one vs. meager comparison of the references.
Overloading the assignment operator in C++

```cpp
class Date {...};
Date today(2015, 01, 07);
Date tomorrow(2015, 01, 08);
today = tomorrow; // Call the assignment operator
```

And, what does the assignment operator do?²

User Defined
Whatever…

- Can anyone really understand programmers’ mind?
- The author of these slides (at least) cannot.

Default Behavior
not so clear…

- recursively apply assignment operator on each of the fields
- Non-user defined types: shallow copy.
- The default assignment operator will typically be shallow copy.

²If you think you are smart, please repeat for copy constructor.
Case study: assignment & copy in **EIFFEL**

**Value Semantic**  Atomic types, such as **Char**, **Integer**, **Real**, and **Boolean**, and object attributes marked as **expanded**

**Reference Semantic**  Everything else

---

**Reference (identity) operations:**
- `a := b`  --- Reference assignment
- `a = b`  --- Reference equality testing
- `a /= b`  --- Reference inequality testing

**Shallow (state) operations:**
- `a.copy(b)`  --- Attribute by attribute shallow copy
- `a := clone(b)`  --- Create new clone object
- `equal(a,b)`  --- Attribute by attribute comparison

**Deep (state) operations:**
- `a.deep_copy(b)`  --- Attribute by attribute copy & cloning of inner objects
- `a := deep_clone(b)`  --- Create a full clone of a complex structure
- `deep_equal(a,b)`  --- Attribute by attribute recursive comparison
More general working knowledge

Language Design Question

Suppose that Pascal had a list type constructor

```pascal
VAR
  primes, odds: list of Integer;
```

What does `primes := odds` mean?

**Reference Copying**
- Inconsistent with arrays, records and primitive types
- Pointers in disguise
- Selective updates to one will affect the other

**Value Copying**
- Natural, but inefficient
- Possible solutions: prohibit selective update (as in Lisp), or lazy copying

---

3 Typical exam question, if you like it phrased this way...
Semantics in some contemporary PLs

Value Semantic  Pascal, Lisp, ML, Prolog
Reference Semantic  Java, Smalltalk
Mixed Semantic  Eiffel, C, C++

Most languages have some kind of a mix:

- In Java, primitive types have value semantic
- There are hacks in Lisp that allow reference semantic
- References in ML allow reference semantic
- Eiffel has expanded types
- C# has “non-nullable” types.

In most cases, a conclusive judgment “value/reference semantic” for an entire language is plain wrong.⁴

⁴ as are some of the sweeping judgments made in this slide...
Assignment semantics is defined by the language design:
- C structures follow value semantics.
- C used to place restrictions on passing structures by value.
- Arrays cannot be assigned.
- Pointers are used to implement reference semantics.
- JAVA follows value semantics for primitive types.

Value semantics may be slower
Reference semantics may lead to sharing problems.
Reference semantics is more expressive.