Variables

James Brucker

Variables

- Most programs work on data.
- The values (data) are stored in memory.
- In our program, we need a way to *refer* to things stored in memory...

  "get the value stored in memory location 0x1A08 and add it to the value in memory location 0x1A20."
- Variables are *names* to refer to things stored in memory.

Declaring a Variable

- You must *declare* a variable before you use it.
- You must declare the type of data the variable refers to.

```
double sum;                // declare 'sum' is a double
int count = 0;             // variable of primitive type
String greet = "hello";
                      // greet refers to String object
```

4 Kinds of Variables

```
class BankAccount {
    private static double rate = 0.05;
    private double balance;


public void deposit(double amount) {
    balance = balance + amount;
}

public void doInterest( ) {
    int minimum = 200;
    if ( balance >= minimum ) {
    double interest = balance * rate;
    balance += interest;
}
```

static attribute of class

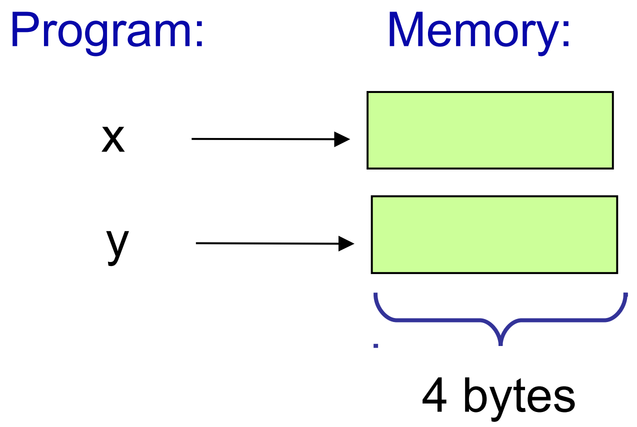
attribute of an object.

parameter to a method, exists while method is running

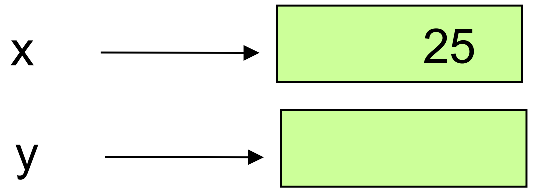
local variable exist while a block is active

Variables and Memory

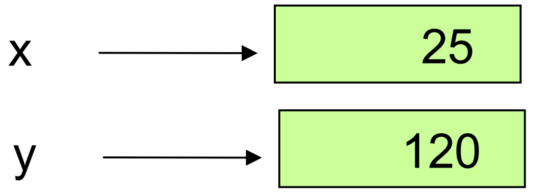
```
/* define two "int"
  variables */
int x;
int y;
```



```
/* assign value to x */
x = 25;
```



```
/* assign value to y */
y = 4 * x + 20;
```



For primitive data types, the memory location of a variable contains its value.

Naming Variables

- First character must be a letter (a-z,A-Z), $ or _ (underscore).
- Followed by any number of letters, numbers, _, or currency symbol.

Valid Variable Names	Invalid Names
`x, money`	`int, public`
`$money`	`final`
`TIME_OUT`	`TIME-OUT`
`one2car`	`1twocar`
`seven11`	`7eleven`
`_value`	`yahoo.com  yahoo!`

Java reserved words

These names are *reserved* in Java. You cannot use any of these words as the name of a variable, label, or class.

`abstract`	`default`	`if`	`private`	`this`
`boolean`	`do`	`implements`	`protected`	`throw`
`break`	`double`	`import`	`public`	`throws`
`byte`	`else`	`instanceof`	`return`	`transient`
`case`	`extends`	`int`	`short`	`try`
`catch`	`final`	`interface`	`static`	`void`
`char`	`finally`	`long`	`strictfp`	`volatile`
`class`	`float`	`native`	`super`	`while`
`const`	`for`	`new`	`switch`	`enum`
`continue`	`goto`	`package`	`synchronized`	

Java doesn't use the words "goto" or "const".

So why are they reserved?

Names are *Case Sensitive*

- Uppercase letters and lowercase letters are distinct!
- This rule applies to all Java syntax

Example:

```
int SUM = 0;
int Sum = 1;
int sum = 2;
```

3 different variables!

Find the Errors

```
// this line has 4 errors:
Public Static Void main( string [] args ) {
   int byte = 0;
   byte = system.in.read( );  // read one byte
   system.out.println('You input ' + Byte );
   System.Exit(0);
}
```

Variable Naming Convention

Please always name variables using these rules:

- first letter is lowercase. First letter of embedded word is Uppercase. Don't use _ between words.

  Good: `accountBalance, topOfList, bestStudent`

  Bad: `AccountBalance, top_of_list`

- use *descriptive* names, avoid abbreviations

  Good: `accountBalance, area, radius`

  Bad: `acctBal, a, r`

Exception: short name is OK for loop index

OK:
```
for( int k=0; k < n; k++ )
    System.out.println( "k = " + k );
```

Java Naming Convention

Makes code *easy to read* and *easier to remember names*

- Java keywords are lowercase
  - "public static", "if", "while", "true", "void
- Primitive datatypes are lowercase
  - boolean, byte, char, double, float, int, long, short
- Class names are Title Case -cap. first letter each word
  - String, System, Math, InputStream, URL
- Wrapper classes are classes, so use Title Case
  - Boolean, Byte, Character, Double, Integer, Long,
- Constants - all UPPERCASE with UNDER_SCORE
  - Math.PI,   Integer.MAX_VALUE, X_AXIS

Example: Correct Use of Names

```
public class BankAccount {
   public static final String ACCT_PREFIX ="11";
   // attributes of a bank account
   private double balance;
   private long accountNumber;
   /** constructor for new accounts */
   public BankAccount( String name, long id ) {
      accountName = name;
      accountNumber = id;
   }
   /** add money to account */
   public void credit( double amount ) {
      balance = balance + amount;
   }
}
```

Example: Wrong Use of Names

```
public class bankaccount {
   // attributes of a bank account
   private String AcctName;
   private double BALANCE;
   private long number;
   /** constructor for new accounts */
   public bankaccount( String n1, long n2 ) {
      AcctName = n1;
      number = n2;
   }
   /** add money to account */
   public void Credit( double a ) {
      BALANCE = BALANCE + a;
   }
}
```

Scope of Variables

- The area of a program where a variable name (or any *identifier*) is known is called the **scope**.
- Each programming language has its own scoping rules. In Java...

Attribute: scope is the entire class, but it may be "shadowed" by a local variable or parameter that has the same name.

Parameter: scope is a method

Local Variable: scope is **from** the point it is declared **to** the end of { ... } block where it is declared.

Scope of Attributes (1)

The scope of an attribute is the entire class, regardless of where the attribute is declared.

```
public class BankAccount {
    private String accountNumber;
    private static long interestRate; // %
    /** create a new bank account object
     */
    public BankAccount( String number,
    String name )  {
    accountNumber = number;
    accountName = name;
    balance = 0;
    }


    // you can define attributes anywhere
    private long balance;
    private String accountName;
    private int homeBranch;
}
```

Scope of Attributes (2)

Inside a method, a local variable or parameter can *shadow* an attribute. In this case, refer to the attribute using scope resolution:

`this.`*`attributeName`*

```
public class BankAccount {
    private String accountNumber;
    private static long interestRate; // %
    private long balance;
    private String accountName;


public void setBalance ( long balance )  {
    // balance parameter shadows
    // balance attribute.

    if ( balance >= 0 )
    this.balance = balance;
    }


}
```

balance *parameter*

balance *attribute*

Common Scope Errors

```
public class BankAccount {
   private String accountNumber;  // attributes
   private String accountName;
   private long balance;

   /** parameterized constructor */
   public BankAccount(String aname, String id) {
      String accountName = aname;
      String accountNumber = id;
      long balance = 0;
   }
   /** a public mutator to set the balance */
   public void setBalance( long balance ) {
      balance = this.balance;
   }
```

This does NOT initialize the attributes.

This does nothing.

Scope of Parameters

The scope of a parameter is the entire method.
A parameter can *shadow* an attribute with the same name.

In Java, a local variable may not have the same name as a parameter.

```
public class BankAccount {
    private String accountNumber;
    private static long interestRate; // %
    private long balance;
    private String accountName;
    /** create a new bank account object
     */
    public BankAccount( String id,
    String accountName )  {
    ... initialize account info ...
    }

    public void setName( long amount ) {
    ... do something ...
    long amount = 0;
```

parameter shadows attribute with same name

Error: defining a local variable with same name as a parameter

Scope of Local Variables

The scope of a local variable is from the point it is defined to the end of the enclosing { ... } block.

```
public class BankAccount {
    private String accountNumber;
    private static long interestRate; // %
    private long balance;
    private String accountName;
    /** create a new bank account object
     */
    public long presentValue( int years,
    long amount ) {
    long pv = amount;
    or(int k = 0; k<years; k++) {
          pv = pv / (1.0 + interestRate);
    }
    // k is undefined here!
    return pv;
    }
```

scope of pv

scope of k

Scope of Local Variables (2)

Error:

```
public double totalData( )   {
    Scanner scan = new Scanner( ... );
    double sum = 0.0;
    while( scan.hasNextDouble( ) ) {
    double x = scan.nextDouble( );
    sum += x;
    }
    System.out.println("Last value was: "
    + x );
    return sum;
}
```

Variables and Values

- A variable of a *primitive type* contains a value of the primitive.
    - Assigning the value to another variable creates a copy of the value.

```
int n = 10;
int m = n;      // copy the value to m
n = 5;          // no effect on m
out.print(m);  // prints 10
```

Variables as References

- A variable of a *class or interface type* contains a reference to an object (which may be null).
  - Assigning the value to another variable makes both variables *refer to the same object*.
  - a = b; copies the *reference*, not the *object.*

```
Date d = new Date( );
Date x = d;       // x refers to same date
d.setYear( 0 ); // change the year
                // this changes x, too!
```