Language Idioms

- Many languages have common “phrases”: *idioms*
- Example: middle-exit loops:

```
while (true)
  { ...
    if (...)
      break;
    ...
  } // while
```

```
for (;;)
  { ...
    if (...)
      break;
    ...
  } // for
```

Chapter 4
Methods and Arguments

Basics

- Methods must appear inside classes
- Heading:
  - return type
  - name
  - parameter list (brackets required)

```
int fact (int n)
  { ...
  } // fact
```

```
int numPixels (int h, int w)
  { ...
  } // numPixels
```

Void Methods

- Methods that do not return any result (procedures, subprograms)
- Can have a **`return`** (without any value)

```
void printTime (int hrs, int min)
  { System.out.print(”Time: ” + hrs + ”:”
                          + min);
  } // printTime
```

Methods

- Cannot be nested
- May call themselves recursively
- May only use value parameters

Method Overloading

- Many methods may have the same name

```
int max (int u, int v)
  { if (u > v)
      return u;
    else
      return v;
  } // max
```

```
char max (char u, char v)
  { if (u > v)
      return u;
    else
      return v;
  } // max
```

```
double max (double u, double v)
  { if (u > v)
      return u;
    else
      return v;
  } // max
```

Method Overloading (cont.)

```
x = max(a, b);
```

- How does the compiler know which method to call?
- By looking at the *number* and the *types* of the actual parameters (i.e. **`a`** and **`b`** above)
  - The return type is ignored
  - Automatic promotions may be applied

Chapter 5
Constants

The `final` Modifier

- Declarations can be marked as **`final`**

```
final double PI = 3.1415926;
```

- Parameters can also be marked as **`final`**
  - A little pointless in Java!

Chapter 6
Classes and Encapsulation

Introduction

- Classes as new data types
- Creating *Abstract Data Types (ADTs)*
- The use of **`static`**

Classes

- Often we need to work with complex data structures

- Examples:
  - student
  - bank account
  - combo box

- These are *aggregate data types*

Classes (cont.)

- A class allows us to define a new type, made up of a number of parts:
  - *data members* (or: *attributes*, or *properties*, or *fields*)
- Represented diagrammatically:

Classes (cont.)

- We distinguish between:
  - defining new classes **Implementors**
  - using such classes **Clients**