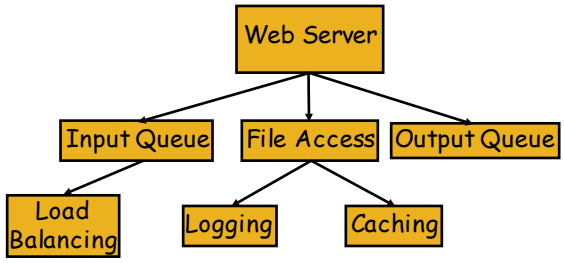
Modules, Abstraction, and Objects

CSCI 334
Stephen Freund

Program Design



Example: power Function

- Interface
  - `fun power : int * int -> int`
- Specification
  - `power(m,n)` returns the value m^n
- Implementation

```
fun power(m,0) = 1
  | power(m,n) = m * power(m,n-1);
```

Example: Stack

- Interface

```
type Stack
val empty : Stack;
fun push : int * Stack -> Stack
fun pop : Stack -> int * Stack
```

- Specification
  - Pre/Post conditions
  - "behaves like a stack..."
- Implementation

```
type Stack = int list;

val empty = nil;

fun push(n, ns) = n::ns;

fun pop(nil) = (0, empty)
  | pop(n::ns) = (n, ns);
```

abstype Stack

```
abstype Stack =
    StackRep of int list
with
   val empty = StackRep(nil);
   fun push(n, StackRep(ns)) = StackRep(n::ns);
   fun pop(StackRep(nil)) = (0,empty)
     | pop(StackRep(n::ns)) = (n,StackRep(ns));
end;

- empty;
val it = - : Stack
- val st = push(3, empty);
val st = - : Stack
- val (top,_) = pop(st);
val it = 3 : int
```

Expr abstype

```
- abstype Expr =
    VarX | Times of Expr * Expr
  with
    fun buildX()             = VarX;
    fun buildTimes(e1,e2)    = Times(e1,e2);
    fun exprToString ... = ...;
    fun eval ... = ...;
  end;

- val e = buildTimes(buildX(), buildX());
val e = - : Expr
- exprToString e;
val it = "x * x": string
```

Object-Oriented Programming [Grady Booch]

- Organize concepts into objects and classes
  - Identify the objects at a given level of abstraction
    - semantics (intended behavior)
    - relationships among the objects
  - Implement these objects
- Iterative process
  - Implement objects by repeating these steps
- Not necessarily top-down
  - "Level of abstraction" could start anywhere
- **Enables extensible systems designs**

Subtyping and Substitutivity

```
class Rectangle {
  private int x,y,w,h;
  void moveTo(int x, int y);
  void setSize(int width, int height);
  void show();
  void hide();
}

class FilledRectangle {
  private int x,y,w,h;
  private Color c;
  void moveTo(int x, int y);
  void setSize(int width, int height);
  void show();
  void hide();
  void setFillColor(Color color);
  Color getFillColor();
}
```

Subtyping and Substitutivity

```
void f() {
  Rectangle r =
    new Rectangle();
  r.moveTo(100,100);
  r.hide();
}

void g() {
  FilledRectangle r =
    new FilledRectangle();
  r.moveTo(100,100);
  r.setFillColor(Color.red);
  r.hide();
}
```

```
void f() {
  Rectangle r =
    new FilledRectangle();
  r.moveTo(100,100);
  r.hide();
}

void g() {
  FilledRectangle r =
    new Rectangle();
  r.moveTo(100,100);
  r.setFillColor(Color.red);
  r.hide();
}
```

Rectangles Revisited

```
class Rectangle {
  int x,y,w,h;
  void moveTo(int x, int y);
  void setSize(int width, int height);
  void show();
  void hide();
}

class FilledRectangle extends Rectangle {
  Color c;
  void setFillColor(Color color);
  Color getFillColor();
}
```

Java Subtyping (Sneak Preview...)

```
interface MouseListener {
  void onMousePress(MouseEvent e);
  void onMouseRelease(MouseEvent e);
  void onMouseMove(MouseEvent e);
  void onMouseClick(MouseEvent e);
  ...
}

class StockTicker extends Applet implements MouseListener {
  ...
  void onMousePress(MouseEvent e) { ... }
  void onMouseRelease(MouseEvent e) { ... }
  void onMouseMove(MouseEvent e) { ... }
  void onMouseClick(MouseEvent e) { ... }
  ...
}
```

OO Program Structure

- Group data and functions
- Class
  - Defines behavior of all objects that are instances of the class
- Subtyping
  - Place similar data in related classes
- Inheritance
  - Avoid reimplementing functions that are already defined

Example: Graphics Library

- Identify general concept: Shape
- Specific shapes:
  - Circle, Square, ...
- Operations on shapes

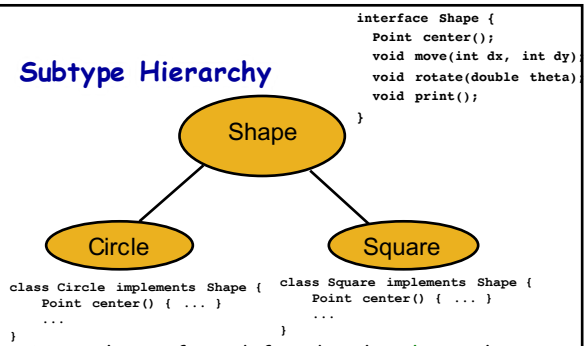
  center, move, rotate, draw
- Anticipate additions to library

Shapes

- Interface of every Shape must include

  center, move, rotate, draw
- Different kinds of shapes have different representations
  - Square: two points, representing corners
  - Circle: center point and radius

Subtype Hierarchy



- General interface defined in the Shape class
- Implementations defined in Circle, Square
- Extend hierarchy with additional shapes

Code Placed In Classes

```
class Circle implements Shape {
  Point center() { ... }
  void move(int dx, int dy) { ... }
    ...
}

class Square implements Shape {
  Point center() { ... }
  void move(int dx, int dy) { ... }
    ...
}
```

- Dynamic lookup
  - `Shape s = new Circle();`
    `s.move(x,y)` calls Circle's move function

Example Use: Processing Loop

```
Vector<Shape>  shapes =
     new Vector<Shape>();
...
for (i = 0; i < shapes.size();  i++)  {
  Shape  s = shapes.get(i);
  s.move(10,  10);
}
```