Announcements

http://recommendz.mcgill.ca

Remarks on constructors

```
class A {
    String s;
    A(String q)
    {
        s = "hello "+q;
    }
}

public class ConsTest {
    public static void main(String[] args)
    {
        A x = new A();  // Error
        System.out.println(x.s);
    }
}
```

Remarks on constructors

```
class A {
    String s;
    A(String q)
    {
        s = "hello "+q;
    }
}

public class ConsTest {
    public static void main(String[] args)
    {
        A x = new A(“bye”);
        System.out.println(x.s);
    }
}
```

Remarks on constructors

```
class A {
    String s;
    A() { s = "bonjour "; }
    A(String q)
    {
        s = "hello "+q;
    }
}

public class ConsTest {
    public static void main(String[] args)
    {
        A x = new A();
        System.out.println(x.s);
    }
}
```

Remarks on constructors

```
class A {
    String s;
    A()
    {
        s = "hello ";
    }
}
class B extends A {
    int n;
}
public class ConsTest {
    public static void main(String[] args)
    {
        B b1 = new B();
        System.out.println(b1.s);
    }
}
```

Remarks on constructors

```
class A {
    String s;
    A(String q)
    {
        s = "ask "+q;
    }
}
class B extends A {
    int n;
}
public class ConsTest
{
    public static void main(String[] args)
    {
        B b1 = new B();
        System.out.println(b1.s);
    }
}
```

Remarks on constructors

```
class A {
    String s;
    A() { s = "hello "; }
}

class B extends A {
    int n;
    B(int i)
    {
        n = i;
    }
}

public class ConsTest {
    public static void main(String[] args)
    {
        B b1 = new B(5);
        System.out.println(b1.s);
    }
}
```

Remarks on constructors

```
class A {
    String s;
    A(String q) { s = "hello "+q; }
}
class B extends A {
    int n;
    B(int i)
    {   // Error: no A()
        n = i;
    }
}
public class ConsTest {
    public static void main(String[] args)
    {
        B b1 = new B(5);
        System.out.println(b1.s);
    }
}
```

Remarks on constructors

```
class A {
    String s;
    A(String q) { s = "hello "+q; }
}
class B extends A {
    int n;
    B(int i)
    {
        super(“bye”);
        n = i;
    }
}
public class ConsTest {
    public static void main(String[] args)
    {
        B b1 = new B(5);
        System.out.println(b1.s);
    }
}
```

Remarks on constructors

```
class A {
    String s;
    A()  { s = "bye "; }
    A(String q) { s = "hello "+q; }
}
class B extends A {
    int n;
    B(int i)
    {
        super("salut");
        n = i;
    }
}
public class ConsTest {
    public static void main(String[] args)
    {
        B b1 = new B(5);
        System.out.println(b1.s);
    }
}
```

Remarks on constructors

- If a class `A` does not have a constructor, then it implicitely has a default constructor with no parameters

  ```
  A() { super(); }
  ```

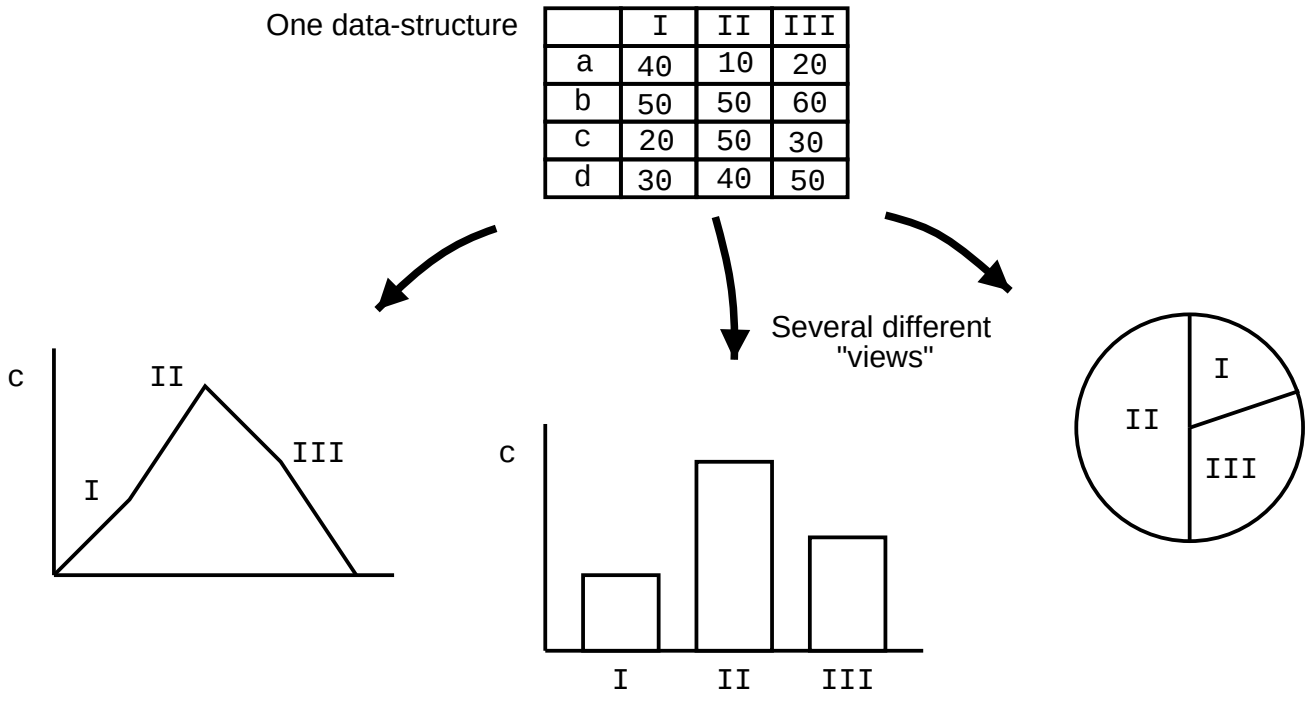
- If a class `A` has a constructor with parameters, then it does not implicitely have a default constructor `A()`

- Constructors are not inherited

- All constructors have an implicit call to the superclass's default constructor, unless it explicitely calls a non-default constructor from the parent.

The "Listener" pattern

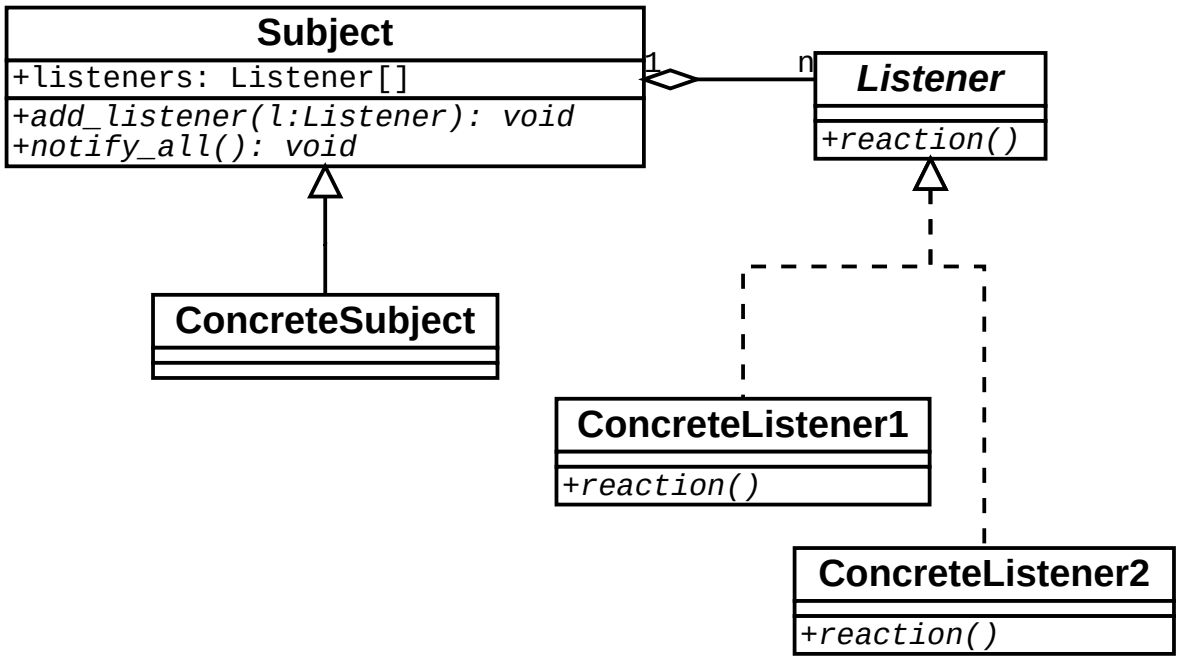
- "Observers", "Callbacks", MVC
- Separation of concerns: data structures and computation should be independent and separated from the user-interface.



	I	II	III
a	40	10	20
b	50	50	60
c	20	50	30
d	30	40	50

- Data-structures, algorithms, and UI should be in separate modules

The "Listener" pattern




```
interface Listener {
  void reaction();
}
```

The "Listener" pattern

```
class Subject {
  Listener[] list;
  int counter;
  Listened()
  {
    list = new Listener[100];
    counter = 0;
  }
  void add_listener(Listener l)
  {
    if (counter < list.length) {
      list[counter] = l;
      counter++;
    }
  }
  void notify_all()
  {
    for (int i=0; i < counter; i++) {
      list[i].reaction();
    }
  }
}
```

The "Listener" pattern

```
class Peeper implements Listener {
  void reaction()
  {
    System.out.println(“I’m watching”);
  }
}

class Eavesdropper implements Listener {
  void reaction()
  {
    System.out.println(“I’m listening”);
  }
}

class Neighbour extends Subject {
  void party() {
    // Do stuff...
    notify_all();
  }
}
```

The "Listener" pattern

```
class BuildingTest {
  public static void main(String[] args)
  {
    Neighbour s = new Neighbour();
    Peeper l1 = new Peeper();
    Eavesdropper l2 = new Eavesdropper();
    Eavesdropper l3 = new Eavesdropper();
    s.add_listener(l1);
    s.add_listener(l2);
    s.add_listener(l3);
    s.party();
  }
}
```

The "Listener" pattern

```
class Spy implements Listener {
  Spy(Subject s)
  {
    s.add_listener(this);
  }
  void reaction()
  {
    System.out.println(“I’m taking pictures”);
  }
}
class BuildingTest {
  public static void main(String[] args)
  {
    Neighbour s = new Neighbour();
    Peeper l1 = new Peeper();
    Eavesdropper l2 = new Eavesdropper();
    Spy jb = new Spy(s);
    s.add_listener(l1);
    s.add_listener(l2);
    s.party();
  }
}
```

Applets

- An applet is an application embedded in a webpage
- An applet takes the form of a Java program whose "main" class extends java.applet.Applet.
- And it is embedded in a webpage with the <applet>...</applet> HTML tag
- Java program (MyApplet.java):

```
import java.applet.Applet;
import java.awt.*;
public class MyApplet extends Applet {
  public void paint(Graphics g)
  {
    g.drawString("Hello", 60, 30);
  }
}
```

Applets

- In any HTML file:

```
<applet code="MyApplet.class" width=300
        height=200>
</applet>
```

- For example:

```
<html>
  <head>
    <title>My web page</title>
  </head>
  <body>
    Here is my applet:
    <applet code="MyApplet.class"
            width=300 height=200></applet>
  </body>
</html>
```

Applets

- The paint method gets executed every time the applet needs to visualize it.

```
public class Myapplet extends Applet {
  public void paint(Graphics g) {
    g.drawRect(50, 70, 40, 20);
    g.drawLine(35, 10, 90, 90);
    g.drawString("Some text here", 50, 70);
    g.drawString("Welcome to Java!!", 50, 60 );
    g.setColor(Color.green);
    g.fillOval(80, 120, 20, 50);
    g.setColor(Color.blue);
    g.fillRect(120,80, 50, 20);
    g.setColor(new Color(20, 200, 100));
    g.fillRect(120, 120, 50, 50);
    setBackground(new Color(175, 175, 75));
  }
}
```

Applets

```
import java.awt.*;
class Circle
{
  int x, y, r;
  Color c;
  Circle(int x, int y, int r)
  {
    this.x = x;
    this.y = y;
    this.r = r;
    c = new Color(75, 175, 175);
  }
  void draw(Graphics g)
  {
    Color current = g.getColor();
    g.setColor(c);
    g.fillOval(x - r, y - r, 2 * r, 2 * r);
    g.setColor(current);
  }
}
```

Applets

```
import java.applet.Applet;
import java.awt.*;
public class Myapplet extends Applet {
  Circle b;
  public void init()
  {
    b = new Circle(200, 200, 40);
  }
  public void start()
  {
  }
  public void stop()
  {
  }
  public void paint(Graphics g) {
    b.draw(g);
  }
}
```

Applets

```
public class MouseEvent {
  public Point getPoint() { ... }
}

public class Point {
  public double getX() { ... }
  public double getY() { ... }
}

public interface MouseListener {
  public void mouseClicked(MouseEvent e);
  public void mousePressed(MouseEvent e);
  public void mouseReleased(MouseEvent e);
  public void mouseEntered(MouseEvent e);
  public void mouseExited(MouseEvent e);
}
```

Applets

```
import java.applet.Applet;
import java.awt.*;
public class Myapplet extends Applet {
  Point p;
  public void init()
  {
    p = null;
    addMouseListener(new MyMouseListener(this));
  }
  public void set_point(Point p)
  {
    this.p = p;
  }
  public void paint(Graphics g) {
    if (p != null) {
      Circle b = new Circle((int)p.getX(),
                            (int)p.getY(), 10);
      b.draw(g);
    }
  }
}
```

Applets

```
import java.awt.event.*;
class MyMouseListener implements MouseListener {
  Myapplet applet;
  MyMouseListener(Myapplet a)
  {
    applet = a;
  }
  public void mouseClicked(MouseEvent e)
  {
    Point p = e.getPoint();
    applet.set_point(p);
    applet.repaint();
  }
  public void mousePressed(MouseEvent e) {}
  public void mouseReleased(MouseEvent e) {}
  public void mouseEntered(MouseEvent e) {}
  public void mouseExited(MouseEvent e) {}
}
```