class MyClass
{
    public int x;
    public int y;
}

class Driver
{
    public static void main(String args[]) 
    {
        MyClass C[];
        C[2].x = 4;
    }
}
class MyClass
{
    public int x;
    public int y;
}

class Driver
{
    public static void main(String args[])
    {
        MyClass C[]; // Space for 1 pointer
        C[2].x = 4;  // C[2] doesn’t exit
    }
}
class MyClass
{
    public int x;
    public int y;
}

class Driver
{
    public static void main(String args[])
    {
        MyClass C[] = new C[5];
        C[2].x = 4;
    }
}
class MyClass
{
    public int x;
    public int y;
}

class Driver
{
    public static void main(String args[])
    {
        MyClass C[] = new C[5];
        C[2].x = 4;  C[2] now exists
        C[2].x does not exist
    }
    C[2] is null!
}
class MyClass
{
    public int x;
    public int y;
}

class Driver
{
    public static void main(String args[])
    {
        MyClass C[] = new C[5];
        for (int i = 0; i < 5; i++)
        {
            C[i] = new MyClass();
        }
        C[2].x = 4;
    }
}
public class Point
{
    public int x;
    public int y;
}

public class Rectangle
{
    public Point upperLeft;
    public Point lowerRight;
}

public class Driver
{
    public static void main(String args[])
    {
        Point p = new Point();
        Rectangle r = new Rectangle();
        p.x = 3
        p.y = 4;
        r.upperLeft.x = 2
        r.upperLeft.y = 5;
    }
}
11-6: Classes in Classes

```java
public class Point {
    public int x;
    public int y;
}

public class Rectangle {
    public Point upperLeft;
    public Point lowerRight;
}

public class Driver {
    public static void main(String args[]) {
        Point p = new Point();
        Rectangle r = new Rectangle();

        p.x = 3; // OK
        p.y = 4; // OK
        r.upperLeft.x = 2; // BAD! Null Pointer
        r.upperLeft.y = 5; // BAD! Null Pointer
    }
}
```
public class Point
{
    public int x;
    public int y;
}

public class Rectangle
{
    public Point upperLeft;
    public Point lowerRight;
}

public class Driver
{
    public static void main(String args[])
    {
        Point p = new Point();
        Rectangle r = new Rectangle();
        p.x = 3 // OK
        p.y = 4; // OK
        r.upperLeft = new Point();
        r.upperLeft.x = 3;
        r.upperLeft.y = 5;
    }
}
Classes in Classes

- Creating a rectangle, and then calling new on each point in the rectangle from the main is awkward.
- With more rectangles, even more of a pain
- What could we do instead?
Creating a rectangle, and then calling new on each point in the rectangle from the main is awkward.

With more rectangles, even more of a pain

What could we do instead?
  • Constructor for Rectangle creates the space that it needs
class Point()
{
    public int x;
    public int y;

    public Point(int x, int y)
    {
        this.x = x;
        this.y = y;
    }
}

in main:

Rectangle r = new Rectangle(3,4,10,12);
class Point() {
    public int x;
    public int y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}

class Rectangle {
    public point upperLeft;
    public point lowerRight;

    public Rectangle(int x1, int y1, int x, int y2) {
        upperLeft = new Point(x1, y1);
        lowerRight = new Point(x2, y2);
    }
}

in main:

Rectangle r = new Rectangle(); OK?
11-12: Classes in Classes

```java
class Point()
{
    public int x;
    public int y;

    public Point(int x, int y)
    {
        this.x = x;
        this.y = y;
    }
}

class Rectangle
{
    public Point upperLeft;
    public Point lowerRight;

    public Rectangle(int x1, int y1, int x, int y2)
    {
        upperLeft = new Point(x1, y1);
        lowerRight = new Point(x2, y2);
    }
}
```

in main:

Rectangle r = new Rectangle();  NOT OK!

- When you define a constructor with parameters, lose the constructors with no parameters
class Point() {
    public int x;
    public int y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}

class Rectangle {
    public point upperLeft;
    public point lowerRight;

    public Rectangle(int x1, int y1, int x, int y2) {
        upperLeft = new Point(x1, y1);
        lowerRight = new Point(x2, y2);
    }

    public Rectangle() {
        upperLeft = new Point(0,0);
        upperRight = new Point(0,0);
    }
}
class Point()
{
  public int x;
  public int y;

  public Point(int x, int y)
  {
    this.x = x;
    this.y = y;
  }
}

class Rectangle
{
  public point upperLeft;
  public point lowerRight;

  public Rectangle(int x1, int y1, int x, int y2)
  {
    upperLeft = new Point(x1, y1);
    lowerRight = new Point(x2, y2);
  }

  public Rectangle()
  {
    this(0,0,0,0); // call the constructor w/ 4 params
    // Must be the 1st line of the constructor
  }
}
11-15: Classes in Classes

class A
{
    public int x;
    public int y;
}
class B
{
    public A class1;
    public int z;
    public A class2;
}
class C
{
    public A class1;
    public int x;
    public B class2;
}
```java
class A {
    public int x;
    public int y;
}

class B {
    public A class1;
    public int z;
    public A class2;
}

class C {
    public A class1;
    public int x;
    public B class2;
}

class Driver {
    public static void main(String args[]) {
        C myC = new C();
        myC.class1 = new A();
        myC.class2 = new B();
        myC.class2.class1 = new A();
        myC.class2.class2 = new A();
    }
}
```
11-17: Classes in Classes

class A
{
    public int x;
    public int y;
}
class C
{
    public A class1;
    public int x;
    public B class2;
}
class B
{
    public A class1;
    public int z;
    public A class2;
}
public C()
{
    class1 = new A();
    class2 = new B();
}
public B()
{
    class1 = new A();
    class2 = new A();
}
}
Getters and Setters

- All of these examples have used public instance variables
  - Mostly so that several classes could fit on one slide
- Usually a very *BAD* idea – giving users too much access to the internal portions of your code
- Instead, make instance variables private (or protected), only access values through method calls
  - Simplest possible: getters and setters, to get and set values of variables
  - Think about your ordered string list: If a user changed the size, strange things would happen. Don’t let them!
11-19: More on Inheritance

class A
{
    public int x;
    public int y;
}
class B extends A
{
    public int w;
}

• Every variable of type B contains all instance variables in A as well
• It’s as if you copy/pasted everything from A into B
class A {
    public int x;
    public int y;
}

class B extends A {
    public int w;
}

B myB = new B();
myB.x = 3;
myB.y = 4;
myB.w = 5;
class A
{
  public int x;
  protected int y;
  private int z;

  void set3(int a, int b, int c)
  {
    x = a;
    y = b;
    z = c;
  }

  void set4(int a, int b, int c, int d)
  {
    x = a;
    y = b;
    z = c;
    w = d;
  }
}

A c1 = new A();
A c2 = new B();
c1.set3(1, 2, 3);
c2.set3(4, 5, 6);
c2.set4(7, 8, 9, 10);
### 11-22: Yet More on Inheritance

```java
class A {
    public int x;
    protected int y;
    private int z;

    void set3(int a, int b, int c) {
        x = a;  // OK
        y = b;  // OK
        z = c;  // BAD (private!)
    }
}

class B extends A {
    private int w;

    void set4(int a, int b, int c, int d) {
        x = a;  // OK
        y = b;  // OK
        z = c;  // BAD (private!)
        w = d;  // OK
    }
}

A c1 = new A();
A c2 = new B();
c1.set3(1, 2, 3);
c2.set3(4, 5, 6);
c2.set4(7, 8, 9, 10);
```

How could we do this?
class A
{
    public int x;
    protected int y;
    private int z;
    
    void set3(int a, int b, int c)
    {
        x = a;
        y = b;
        z = c;
    }
}

A c1 = new A();
A c2 = new B();
c1.set3(1, 2, 3);
c2.set3(4, 5, 6);
c2.set4(7, 8, 9, 10);

class B extends A
{
    private int w;
    
    void set4(int a, int b, int c, int d)
    {
        set3(a, b, c);
        w = d;
    }
}
class A
{
  public int x;
}

class B extends A
{
  public int d;
}

In main:
--------
A a;
B b;

a.x = 3;  OK?
b.x = 4;  OK?
b.d = 5;  OK?
class A
{
    public int x;
}

class B extends A
{
    public int d;
}

In main:
--------
A a;
B b;

a.x = 3; NOT OK -- cannot use class variable until "new" is called
All class variables (and arrays!) are stored on the heap
b.x = 4; NOT OK
b.d = 5; NOT OK
class A
{
    public int x;
    public C c;
    public int intArray[];
    public C[] cArray;
}

class B extends A
{
    public int d;
}

class C
{
    public int e;
}
class A
{
    public int x;
    public C c;
    public int intArray[];
    public C[] cArray;
}

class B extends A
{
    public int d;
}

class C
{
    public int e;
}

In Main
---------
A a = new A();
B b = new B();
a.x = 3;  OK
a.c.e = 4;  NOT OK
b.x = 5;  OK
b.d = 6;  OK
a.intArray[2] = 4;  NOT OK
class A
{
    public int x;
    public C c;
    public int intArray[];
    public C[] cArray;
}

class B extends A
{
    public int d;
}

class C
{
    public int e;
}
class A
{
    public int x;
    public C c;
    public int intArray[];
    public C[] cArray;
}

class B extends A
{
    public int d;
}

class C
{
    public int e;
}
class A
{
    public int x;
    public C c;
    public int intArray[];
    public C[] cArray;
}

In Main
--------
A a[] = new A[2];
int x;
int y[] = new int[3];
a[2].x = 4
a[2].cArray = new C[3];
a[2].cArray[2].e = 4
a.x = 3
x = 3;
y[3] = 4;

class B extends A
{
    public int d;
}

class C
{
    public int e;
}
class A
{
    public int x;
    public C c;
    public int intArray[];
    public C[] cArray;
}

class B extends A
{
    public int d;
}

class C
{
    public int e;
}

In Main
---------
A a[] = new A[2];
int x;
int y[] = new int[3];
a[2].x = 4
a[2].cArray = new C[3];
a[2].cArray[2].e = 4
a.x = 3
x = 3;
y[3] = 4;

NOT OK
NOT OK
NOT OK
OK
OK
class A
{
    public int x;
    public C c;
    public int intArray[];
    public C[] cArray;
}

class B extends A
{
    public int d;
}

class C
{
    public int e;
}

In Main
--------
A a[] = new A[3];
a[0] = new B();  OK?
a[1] = new A();  OK?
a[2] = new B();  OK?
a[3] = new A();  OK?
a[0].x = 2;        OK?
a[a[0].x].x = 4  OK?
a[2].e = 5;        OK?
a[2].c = new C(); OK?
a[2].c.e = 6;     OK?
11-33: Yet More on Inheritance

class A
{
    public int x;
    public C c;
    public int intArray[];
    public C[] cArray;
}

class B extends A
{
    public int d;
}

class C
{
    public int e;
}

In Main
---------
A a[] = new A[3];
a[0] = new B();    OK
a[1] = new A();    OK
a[2] = new B();    OK
a[3] = new A();    OK
a[0].x = 2;        OK
a[0].x = 4;        OK
a[2].e = 5;        NOT OK
a[2].c = new C();  OK
a[2].c.e = 6;      OK
class A
{
    public int x; // Note on public instance vars
    public int y;
}
class B
{
    public int w;
    public A aInstance;
}
class C extends B
{
    public int v;
    public A anotherAInstance;
}
(in main)
C cArray[] = new C[10]; // OK?
cArray[2].v = 3 // OK?   cArray[2].anotherAInstance.x = 3; // OK?
cArray[2].w = 4 // OK?   cArray[2].aInstance = new A(); // OK?
class A
{
    public int x; // Note on public instance vars
    public int y;
}
class B
{
    public int w;
    public A aInstance;
}
class C extends B
{
    public int v;
    public A anotherAInstance;
}
(in main)
C cArray[] = new C[10]; // OK!
cArray[2].v = 3 // BAD cArray[2].anotherAInstance.x = 3; // BAD
cArray[2].w = 4 // BAD
    cArray[2].aInstance = new A(); // BAD
class A
{
    public int x; // Note on public instance vars
    public int y;
}
class B
{
    public int w;
    public A aInstance;
}
class C extends B
{
    public int v;
    public A anotherAInstance;
}
(in main)
C cArray[] = new C[10]; // OK!
C cArray2[2] = new C(); // OK!
cArray[2].v = 3 // OK! cArray[2].anotherAInstance.x = 3; // BAD
cArray[2].w = 4 // OK!
cArray[2].aInstance = new A(); // OK!
11-37: **Constructors to the Rescue**

- Constructor calls “new” on instance variables that are not primitive types
- Classes that contain classes that contain classes that contain arrays of classes just work
class A
{
    public int x;
    public int y;
}
class B
{
    public int w;
    public A aInstance;
}

class C extends B
{
    super();
    anotherAInstance = new A();
}
cArray[] = new C[10];
for (int i = 0; i < 10; i++)
    cArray[i] = new C();

// All instance variables properly created