Objects

- Object methods
- Encapsulation
- Permissions
- Constructors
- Equality

See Also

https://docs.oracle.com/javase/tutorial/java/javaOO/methods.html

http://www.tutorialspoint.com/java/java_methods.htm

https://howtoprogramwithjava.com/what-is-a-method-in-java/

Data with Functions

- Related functions that work on a particular structure
- Pass the pointer as the first argument

```
public class Tinker {

    public static void initPoint(Point t, int a, int b) {
        t.x = a;
        t.y = b;
    }

    public static void printPoint(Point t) {
        System.out.println(t.x+","+t.y);
    }

    public static void main(String [] args) {

        Point a = new Point();
        initPoint(a,1,2);

        printPoint(a);
    }

}
```

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

Data with Functions

- Every data structure usually has a set of functions that work on it

```
public class Tinker {

    public static void initLine(Line this, Point v, Point w) {
        this.a = v;
        this.b = w;
    }

    public static void main(String [] args) {

        Point a = new Point();
        initPoint(a,1,2);

        Point b = new Point();
        initPoint(b,3,4);

        Line c = new Line();
        initLine(c,a,b);
    }

}
```

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

```
class Line {
    Point a;
    Point b;
}
```

Together at Last

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

- Keep the functions with the data they work on

```
void initPoint(Point this, int a, int b) {
    this.x = a;
    this.y = b;
}
```

```
void printPoint(Point this) {
    System.out.println(this.x+","+this.y);
}
```

Together at Last

```
class Point {
    int x;
    int y;

    void initPoint(  Point this,   int a, int b) {
             x = a;
             y = b;
    }

    void printPoint(  Point this  ) {
        System.out.println(     x+","+     y);
    }

}
```

- The compiler puts in "this" automatically (you can't)
- The compiler will insert "this." where needed
- You can still use "this." too. You might even need it in some situations

Invoking Methods

- There is a new way to call these functions-linked-to-data
- Move the pointer to the front
- We say we are "invoking a method on an object"
- This is OO – asking an object to do something

```
Point a = new Point();

initPoint( a ,  1,2);

printPoint( a );       a.printPoint();
```

- As if these data structures have code with them
- We follow-the-pointer to code just as to data

OO Concept: Encapsulation

- We have created a "time" structure
- We use it in our Alarm object
- Everybody likes it!

Time.java

```
class Time {
    int hours;
    int minutes;
    int seconds;
}
```

Alarm.java

```
Time t = new Time();
int h = t.hours;
int m = t.minutes;
```

TimeCard.java

```
Time t = new Time();
int h = t.hours;
int m = t.minutes;
```

Replay.java

```
Time t = new Time();
int h = t.hours;
int m = t.minutes;
```

OpenH.java

```
Time t = new Time();
int h = t.hours;
int m = t.minutes;
```

OO Concept: Encapsulation

- Time passes. Our hardware changes. It is better for us to keep time has H*120+M*60+S.
- We change our code to use the new scheme
- What about the other code?

Time.java

```
class Time {
    int hours;
    int minutes;
    int seconds;
}
```

Alarm.java

```
Time t = new Time();
int h = t.hms/120;
int m = (t.hms/60)%60;
```

TimeCard.java

```
Time t = new Time();
int h = t.hours;
int m = t.minutes;
```

Replay.java

```
Time t = new Time();
int h = t.hours;
int m = t.minutes;
```

OpenH.java

```
Time t = new Time();
int h = t.hours;
int m = t.minutes;
```

OO Concept: Encapsulation

- Hide the data behind methods
- Others don't look at the watch directly – they ask the object for the time
- The insides can change as needed

Time.java

```
class Time {

    int hours;
    int minutes;
    int seconds;

    int getHours() {
        return hours;
    }

    int getMinutes() {
        return minutes;
    }

}
```

Time.java

```
class Time {

    int hms;

    int getHours() {
        return hms/60;
    }

    int getMinutes() {
        return (hms/60)%60;
    }

}
```

Alarm.java

```
Time t = new Time();
int h = t.getHours();
int m = t.getMinutes();
```

TimeCard.java

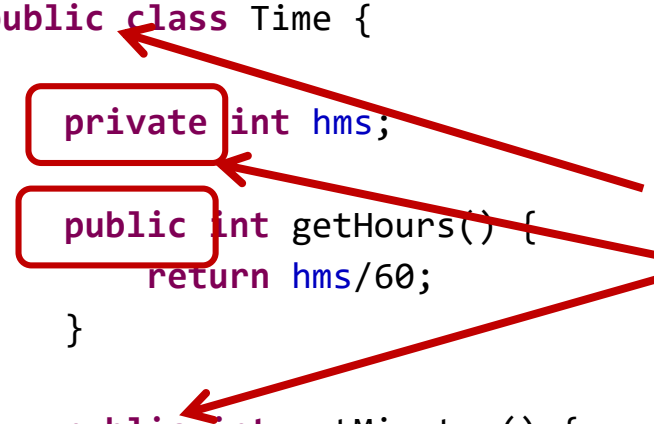
```
Time t = new Time();
int h = t.getHours();
int m = t.getMinutes();
```

Permissions

- "private" means only methods in the class can use
- "public" means everyone can use data or methods
- Create private data with a public interface

Time.java

```
public class Time {

    private int hms;

    public int getHours() {
        return hms/60;
    }

    public int getMinutes() {
        return (hms/60)%60;
    }

}
```



- "" (no keyword) means default permission (permission to classes in the same directory)
- "protected" grants permission to derived classes (later)

- Classes can be "public" or "" (default)
- Data/methods can be "public", "private", "", or "protected"

Constructors

- Creation is a special time. You may want to provide data used to initialize an object.
- Our "init" could be called anytime
- Special method called with "new"

```
public class Point {

    private int x;
    private int y;

    public Point(int a, int b) {
        x = a;
        y = b;
    }

    public void print() {
        System.out.println(x+","+y);
    }

}
```

- No return type on a constructor
- You ALWAYS have a constructor. If you don't provide one the compiler will insert a no-args constructor.

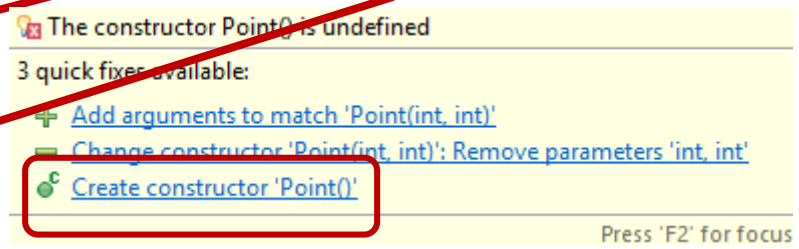
```
Point p = new Point(1,2);
```

Constructors in Action

- If you provide a constructor, the compiler will NOT insert the default.

```
public class Point {

    private int x;
    private int y;

    public Point(int a, int b) {
        x = a;
        y = b;
    }

    public void print() {
        System.out.println(x+","+y);
    }

}
```

```
Point q = new Point(1,2);

Point p = new Point();
```



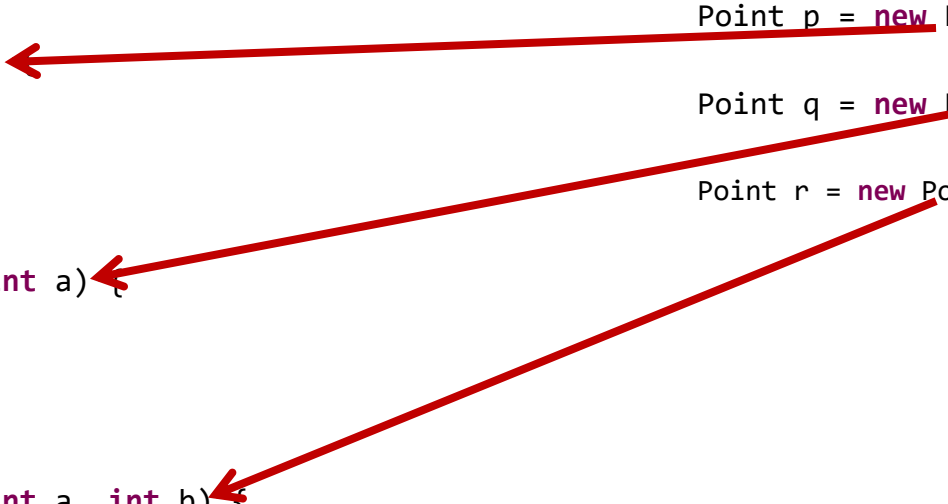
?

Constructors in Action

- You can have multiple constructors – each with a different set of parameters.

```
public class Point {

    private int x;
    private int y;

    public Point()
        x=0;
        y=0;
    }

    public Point(int a) {
        x=a;
        y=a;
    }

    public Point(int a, int b) {
        x = a;
        y = b;
    }

}
```

```
Point p = new Point();

Point q = new Point(3);

Point r = new Point(4,10);
```

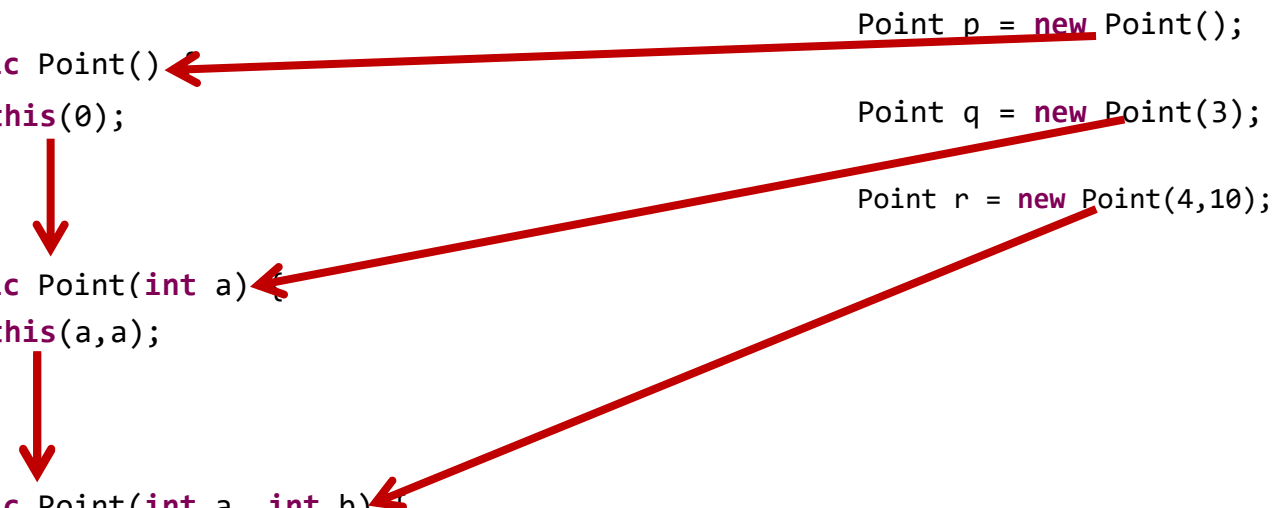


Constructors in Action

- Constructors can call constructors.

```
public class Point {

    private int x;
    private int y;

    public Point() {
        this(0);
    }

    public Point(int a) {
        this(a,a);
    }

    public Point(int a, int b) {
        x = a;
        y = b;
    }

}
```

```
Point p = new Point();

Point q = new Point(3);

Point r = new Point(4,10);
```



Initialization Inline

- Complex initialization must be done in constructor
- Simple inits can be done inline

```
class MyStuff {

    int a; // At "new", initialized to "0"

    int b = 20; // At "new", initialized to "20"

    Point c; // At "new", initialized to "0" (null)

    Point d = new Point(10,20); // At "new", creates a new object

}
```

Object Equality

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

public class Tinker {

    static boolean isTheSameAs(Point a, Point b) {
        if(b.x != a.x) return false;
        if(b.y != a.y) return false;
        return true;
    }

    public static void main(String [] args) {
        Point p = new Point();
        Point q = new Point();

        if(p==q) {}  // Just compares pointers

        if(isTheSameAs(p,q)) {
            // Same coordinates!
        }
    }
}
```

- We might want a "deeper" compare
- In "functional" programming we make a function that looks into the data structures directly

Object Equality

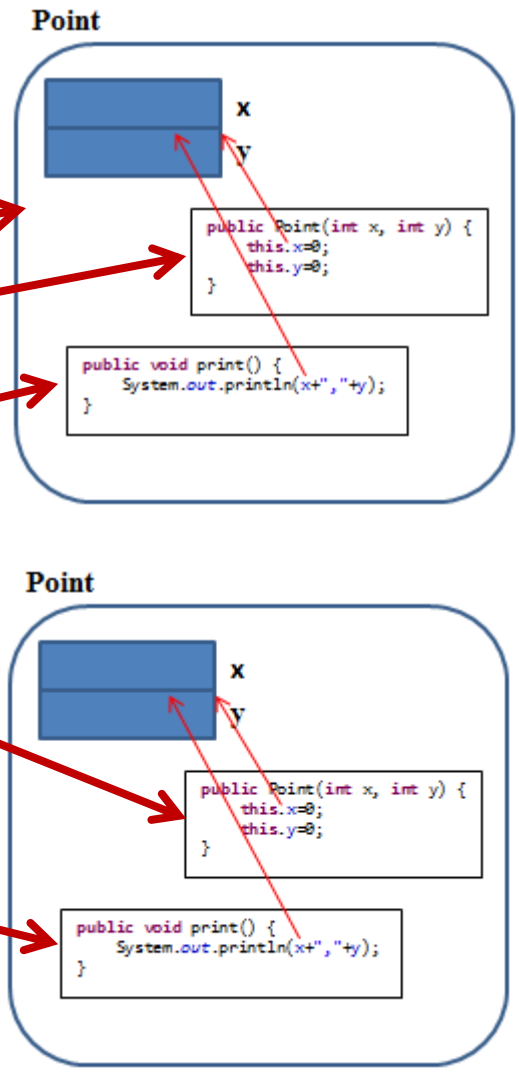
```
public class Point {

    private int x;
    private int y;
}

public class Tinker {

    public boolean isTheSameAs(/*Point this,*/ Point other) {
        if(other.x != this.x) return false;
        if(other.y != this.y) return false;
        return true;
    }

    public static void main(String [] args) {
        Point p = new Point();
        Point q = new Point();

        if(p==q) {}  // Just compares pointers

        if(p.isTheSameAs(q)) {
           // Same coordinates!
        }
    }
}
```

- In OO we keep the code in the data structure and hide the data itself
- We ask one object to compare itself to another

Useful Visualization

- Think of code living in the memory footprint of the object
- Each object has its own copy of the code that works with its own data

```
Point p = new Point(1,2);

Point q = new Point(3,4);

p.print();

q.print();
```



Benefit and Cost

- PRO: Hiding your data makes it easy to change the details in the future. The user doesn't have to worry with the nasty details.

- CON: Hiding data behind a method makes your code slower (you have to make a routine call).

- Is the benefit worth the cost?
- To decide you need:
  - Some experience and intuition
  - A crystal ball to see the future

Your Turn

- Modify your Point.java to include the "print" method and one or more constructors.
- What else might you ask a point to do? Add some more:
  - Calculate distance to another point
  - Set-methods to change X and Y
- Add the "equals" method.
- Make a "main" and create some points. Try calling all the methods you put in the Point.
- What happens if you make your constructors "private"?