

Panel 1

Last time:

Classes: - blueprints to define new types
 = fields (store data) + methods (to do stuff)
 - constructor (class name, no return type)

Objects: - instantiated from classes via "new"
 - they exist in memory for real
 - calls constructor

1

Panel 2

Data Encapsulation: protect integrity of a class

- * private: only objects of same class can access
- * public: everyone can access
- * protected: } everyone in same package or
- none (friendly): } directory has access

Only applies to fields / methods, not local vars

Rule: fields should be private
 methods could be public
 be as restrictive as possible

2

Panel 3

class Temperature

(fields) has: value and unit (100°C or 100K)

do: convert to C, K, F

add / subtract / display

ensure integrity of Temp.

```
class Temperature
```

```
{
```

```
    double value
```

```
    char unit
```

```
    Temperature(double value, char unit)
```

```
    Temperature to F()
```

```
    add()
```

```
}
```

3

Panel 4

```
public class Temperature
```

```
{
```

```
    private double value = 0.0;
```

```
    private double unit = 'C';
```

```
⇒ public Temperature(double value, double unit)
```

```
    public Temperature to F() (etc)
```

```
    public Temperature add (Temperature t) (etc)
```

```
}
```

4

Panel 5

Constructor

```

public Temperature (double value, char unit)
{
    this.value = value;
    if ((unit == 'F') || (unit == 'f'))
        this.unit = 'F';
    else if ((unit == 'K') || (unit == 'k'))
        this.unit = 'K';
    else if ((unit == 'C') || (unit == 'c'))
        this.unit = 'C';
    else
        this.unit = 'I';
}

```

Annotations: 'field' points to 'this.value', 'local' points to 'value'.

Panel 6

```

public void display()
{
    if (unit != 'I')
        System.out.println(value + " " + unit);
    else
        System.out.println("invalid temp");
}

```

Poorly designed method because it assumes text-based output, instead

```

public String display() ← use "to String"
{
    return -----
}
return

```

Panel 7

① Write program to convert

100°C to K, F, C

② Add "add" and "subtract" method at home!

Say I decide to: add always returns temp in $^{\circ}\text{C}$

`t.add(+2)` to Fahrenheit();

③ Really want to show output in
text holds

7

Panel 8

The "static" modifier

fields (and methods) can be static or not.

if non-static, every object gets its own copy of
the field

if static, every object shares the same field

static methods can only refer to static data

Rule of thumb: static method gets all input
from input variables or local
variables.

8

Panel 9

```

public class Temperature
{
    double value;

    public Temperature to F()
    {
        // returns to fields
    }

    public static double convertTo Fahrenheit (double value)
    {
        // convert input value, not field.
    }
}

```

9

Panel 10

Static methods are often "computational".
 They can be used without "new" but by giving class Name.

Ex: double x = Math.sin(Math.PI);

class method (static public)

Commonly used static methods:

Math.sin(#), Math.cos(#), ...

JOptionPane.showMessageDialog (null, " ");

JOptionPane.showInputDialog (" ");

10

Panel 11

```
import javax.swing.JOptionPane;

public class TempTest
{
    public static void main(String args[])
    {
        JOptionPane.showInputDialog("Enter temperature");

        Temperature t = new Temperature(100.0, 'C');

        String s = t.toString() + "\n";
        s += "    = " + t.toFahrenheit().toString() + "\n";
        s += "    = " + t.toKelvin().toString() + "\n";
        s += "    = " + t.toCelsius().toString() + "\n";
        JOptionPane.showMessageDialog(null, s);
    }
}
```

11