

Why objects?

COMP112: 191

- A program has a collection of classes
- Each class has a collection of methods
 - FlagDrawer class had several methods:
 - `public void doJapanFlag()`
 - `public void doFrenchFlag()`
- Why do you have to create a FlagDrawer object before you can call these methods on it?
- Why do you have to call the method on an object?
- What is the object for?

?

© Peter Andreae

Classes and Objects

COMP112: 192

- A class is a description of a type of object.
 - includes descriptions of methods you can call on this kind of object

- Some kinds of objects we have used:

UI

println... ask... next...
draw... fill... clear...

Scanner

next, nextInt, hasNext,...

String

length(), substring...

File

exists...

CartoonFigure

boilWater, toast, bake ...

Flower

grow, bloom, pick ...

- What else did the objects need?
 - Information/Data, specifying the state of the object.
 - Stored in **fields** of the object

© Peter Andreae

What is an Object

COMP112: 193

An object is

- A collection of data wrapped up together

plus

- A collection of actions to operate on the collection of data

All specified in a class:

- Fields where data is stored
- Methods describing the actions
- Constructor to make new objects
- Constants

- Some objects (top level program objects) may have no data.

© Peter Andreae

CartoonStory program

COMP112: 194

- Java Program with 2D cartoon objects
- Uses CartoonCharacter objects:
 - Methods:
 - `public void lookLeft()`
 - `public void lookRight()`
 - `public void smile()`
 - `public void frown()`
 - `public void walk(double distance)`
 - `public void speak(String msg)`
 - `public void think(String msg)`
 - Information a CartoonCharacter object must store:
 - its images
 - its size
 - its state (position, direction, emotion)

© Peter Andreae

CartoonStory Program

COMP112: 195

```
public class CartoonStory{
    public void animate( ){
        CartoonCharacter cf1 = new CartoonCharacter(150, 100, "green");
        cf1.lookRight();
        cf1.lookLeft( );
        cf1.frown( )
        cf1.speak("Is anyone here?");
        CartoonCharacter cf2 = new CartoonCharacter(300, 100, "blue");
        cf2.smile( );   cf2.lookLeft( );   cf2.speak("Hello");
        cf1.lookRight( );   cf1.smile( );
        cf1.speak("Hi there, I'm Jim");
        cf2.speak("I'm Jan");
    }
}
```

Two different objects of the same type

© Peter Andreae

Defining a class of objects

COMP112: 196

- CartoonCharacter is not part of the Java libraries
⇒ have to define the class
- Need to define:
 - methods:
 - specify the actions the objects can do
 - constructor:
 - specifies how to make a new CartoonCharacter object
 - **fields**:
 - for storing the information about the state of each object

© Peter Andreae

CartoonCharacter: methods

COMP112: 197

```
public class CartoonCharacter {
    public void lookLeft( ) {
        // erase figure
        // change direction
        // redraw figure
    }
    public void lookRight( ) {
        // erase figure
        // change direction
        // redraw figure
    }
    public void frown( ) {
        // erase figure
        // change emotion
        // redraw figure
    }
    public void smile( ) {
        // erase figure
        // change emotion
        // redraw figure
    }
    public void walk(double dist) {
        // erase figure
        // change position
        // redraw figure
    }
    public void speak(String msg) {
        // draw msg in bubble
        // wait
        // erase msg
    }
}
```

© Peter Andreae

CartoonCharacter: wishful methods

COMP112: 198

```
public class CartoonCharacter {
    public void lookLeft( ) {
        this.erase( );
        // change direction
        this.draw( );
    }
    public void lookRight( ) {
        this.erase( );
        // change direction
        this.draw( );
    }
    public void frown( ) {
        this.erase( );
        // change emotion
        this.draw( );
    }
    public void smile( ) {
        this.erase( );
        // change emotion
        this.draw( );
    }
    public void walk(double dist) {
        this.erase( );
        // change position
        this.draw( );
    }
    public void speak(String msg) {
        // draw msg in bubble
        // wait
        // erase msg
    }
    public void erase( ) {
        ???
    }
    public void draw( ) {
        ???
    }
}
```

© Peter Andreae

CartoonCharacter: draw

COMP112: 199

```
public void draw() {  
    // work out which image to use (eg, "green/right-smile.png")  
    // draw the image on the graphics pane  
    // wait a bit  
}
```

```
public void draw() {  
    String filename = imageFolder + "/" + direction + "-" + emotion + ".png";  
    UI.drawImage(filename, figX, figY, wd, ht);  
    UI.sleep(500); // wait 500 mS  
}
```

- But where are those variables defined?
- Where do they get their values?

© Peter Andreae

CartoonCharacter Objects

COMP112: 201

- Objects need places to store values – called “Fields”

CartoonCharacter-24	
figX:	<input type="text"/>
figY:	<input type="text"/>
emotion:	<input type="text"/>
direction:	<input type="text"/>
imageFolder:	<input type="text"/>
wd:	<input type="text"/>
ht:	<input type="text"/>

CartoonCharacter-27	
figX:	<input type="text"/>
figY:	<input type="text"/>
emotion:	<input type="text"/>
direction:	<input type="text"/>
imageFolder:	<input type="text"/>
wd:	<input type="text"/>
ht:	<input type="text"/>

- Objects are like entries in your Contacts

© Peter Andreae

Using fields:

COMP112: 202

A method can refer to a field of the object it was called on:

eg: `this.fieldname`

```
public void lookLeft() {  
    this.erase();  
    this.direction = "left";  
    this.draw();  
}
```

note: fields have no ()

Object the method
was called on

```
public void draw() {  
    String filename = this.imageFolder + "/" + this.direction + "-" +  
        this.emotion + ".png";  
    UI.drawImage(filename, this.figX, this.figY, this.wd, this.ht);  
    UI.sleep(500); // wait 500 mS  
}
```

© Peter Andreae

Using fields:

COMP112: 203

Object

CartoonCharacter-24

figX:	<input type="text"/>	wd:	<input type="text"/>
figY:	<input type="text"/>	ht:	<input type="text"/>
emotion:	<input type="text"/>	imageFolder:	<input type="text"/>
direction:	<input type="text"/>		

cf1. lookLeft();
cf1. walk(20);

cf1: CartoonCharacter-24

ID of Object

Method worksheet

```
public void lookLeft() {  
    this.erase();  
    this.direction = "left";  
    this.draw();  
}
```

this: CartoonCharacter-

© Peter Andreae

Using fields:

COMP112: 204

Object

CartoonCharacter-24

figX: 150 wd: 40
figY: 300 ht: 80
emotion: "smile" imageFolder: "green"
direction: "left"

Method Worksheet

```
public void draw() {           this: CartoonCharacter-
    String filename = this.imageFolder + "/" + this.direction + "-" +
    "                    " this.emotion + ".png" ;
    UI.drawImage(filename, this.figX, this.figY, this.wd, this.ht);
    UI.sleep(500);
}
```

© Peter Andreae

Using fields:

COMP112: 205

Object

CartoonCharacter-24

figX: 150 wd: 40
figY: 300 ht: 80
emotion: "smile" imageFolder: "green"
direction: "left"

✓ cfg1.lookLeft();
cfg1.walk(20);
: cfg1: CartoonCharacter-24 ID of Object

```
public void lookLeft() {           this: CartoonCharacter-
    ✓ this.erase() ;
    ✓ this.direction = "left";
    ✓ this.draw() ;
}
```

© Peter Andreae

Using fields:

COMP112: 206

CartoonCharacter-24

figX: 150 wd: 40
figY: 300 ht: 80
emotion: "smile" imageFolder: "green"
direction: "left"

✓ cfg1.lookLeft();
cfg1.walk(20);
:

```
public void walk (double dist) {           this: CartoonCharacter-
    this.erase() ;
    if ( this.direction.equals("right") ) { this.figX = this.figX + dist ; }
    else { this.figX = this.figX - dist ; }
    this.draw() ;
}
```

© Peter Andreae

Objects and Classes

COMP112: 207

Classes define objects:

- Fields: places in an object that store the information associated with the object
methods can refer to fields of the object they were called on:
`this.fieldname`
How do you set up the fields?
- Methods: can be called on any object of the class
- Constructors: specify how to set up an object when it is first created.
- Constants: specify names for values

© Peter Andreae

Setting up an object

COMP112: 208

Must declare the Fields of an object?

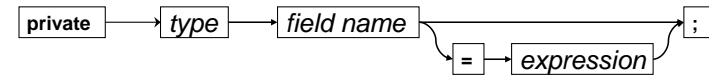
- Declared in the class
(not inside a method)
- Must specify the type and the name
(just like local variables in methods)
- Can specify an initial value (but you don't have to!)
if not, automatically initialised with 0 or null
(unlike local variables)
- Have a visibility specifier ("**private**")
- Fields remain indefinitely
(unlike local variables)
- The set of field declarations is a template for the object
(just like a method is a template for a worksheet).

Just as local variables
must be declared

© Peter Andreae

Syntax of Field declarations:

COMP112: 209



```
public class CartoonCharacter {
    // fields
    private double figX;           // current position of figure
    private double figY;
    private String direction = "right"; // current direction it is facing
    private String emotion = "smiling"; // current emotion
    private String imageFolder;    // base name of images
    private double wd = 40;        // dimensions of figure
    private double ht=80;

    // methods .....
}
```

Like variables, BUT
(a) NOT inside a method
(b) have **private** in front

© Peter Andreae

Setting up an object

COMP112: 210

- How do you initialise the values in the fields?
 - Can specify an initial value in the field declaration
but only if every object should start with the same value!!!
- Must have a way of setting up *different* objects when you create them:

Constructor:

- specifies what happens when you make a new object
(eg, evaluate the expression
new CartoonCharacter(150, 100, "green")
- We have seen constructors with no parameters.
- Can have parameters that can be used to set up the new object.

© Peter Andreae

CartoonCharacter class

COMP112: 211

```
public class CartoonCharacter {
    // fields
    private double figX, figY; // current position of figure
    private String direction = "right"; // current direction it is facing
    private String emotion = "smile"; // current emotion
    private String imageFolder; // folder where images stored
    private double wd = 40, ht=80; // dimensions

    // constructor
    public CartoonCharacter(double x, double y, String base){
        this.imageFolder = base;
        this.figX = x;
        this.figY = y;
        this.draw();
    }

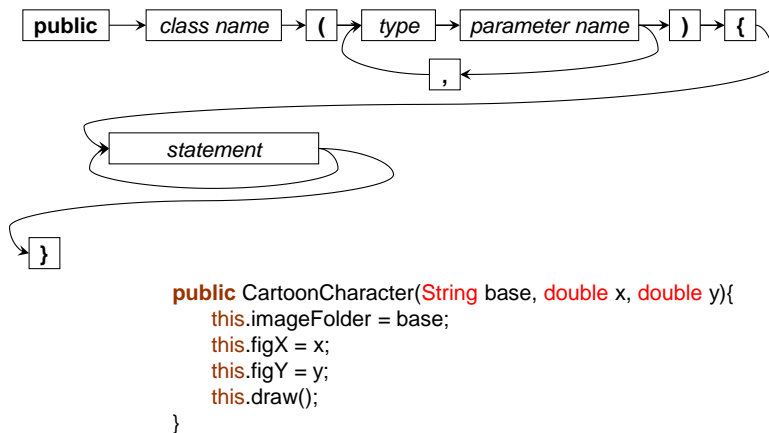
    // methods .....
    public void lookLeft() {
        this.erase(); .....
    }
}
```

Shorthand for declaring two fields
(or variables) of the same type

© Peter Andreae

Syntax of Constructor Definitions (2)

COMP112: 212



© Peter Andreae

Constructors

COMP112: 213

- Defining a Constructor
 - Part of the class
 - Like a method, but called with **new**
 - Does not have a return type
(**new** always returns an object of the given type)
 - **this** will hold the new object that is being constructed
- Constructor typically
 - fills in initial values of fields
 - may call other methods on the object,
 - can do anything an ordinary method can do.
 - The constructor of the "top level" class may set up the user interface.

© Peter Andreae

What happens with **new** ?

COMP112: 214

When an object is created

eg **new** **CartoonCharacter**(100, 200, "yellow");

- New chunk of memory is allocated (new filing card).
- Reference (ID) to object is constructed
CartoonCharacter-24
- Any initial values specified in the field declarations are assigned to the fields.
If no initial value, default values:
 - 0 for fields of a number type (int, double, etc)
 - false for boolean fields
 - null for fields of an object type (String, Scanner, Car, ...)
- The arguments are passed to the constructor
- The actions specified in the constructor are performed on the object.
- The reference is returned as the value of the constructor.

CartoonCharacter-24

figX:	<input type="text"/>
figY:	<input type="text"/>
emotion:	<input type="text"/>
direction:	<input type="text"/>
imageFolder:	<input type="text"/>
wd:	<input type="text"/>
ht:	<input type="text"/>



© Peter Andreae

The whole Program

COMP112: 215

```
public class CartoonStory{
    public CartoonStory(){
        UI.addButton("go", this::playStory);
    }
}
```

```
public void playStory(){
    CartoonCharacter cf1 = new CartoonCharacter(150, 100, "green");
    cf1.lookLeft();
    cf1.lookRight();
    cf1.frown();
    cf1.speak("Is anyone here?");
    CartoonCharacter cf2 = new CartoonCharacter(300, 100, "blue");
    cf2.speak("Hello");
    cf2.lookLeft();
    cf1.smile();
    cf1.speak("Hi there, I'm Jim");
    cf2.speak("I'm Jan");
}
```

```
public static void main(String[] args){
    CartoonStory cs = new CartoonStory();
}
```

Simple class:
- no fields
- constructor for UI
- methods

Note the main method
⇒ don't need BlueJ

© Peter Andreae

CartoonCharacter: fields & constructor

COMP112: 216

```
public class CartoonCharacter {
    // fields
    private double figX;           // current position of figure
    private double figY;
    private String direction = "right"; // current direction it is facing
    private String emotion = "smile"; // current emotion
    private String imageFolder; // base name of image set
    private double wd = 40; // dimensions
    private double ht=80;

    // constructor
    public CartoonCharacter(double x, double y, String base){
        this.imageFolder = base;
        this.figX = x;
        this.figY = y;
        this.draw();
    }
}
```

© Peter Andreae

CartoonCharacter: methods

COMP112: 217

```
public void lookLeft() {
    this.erase();
    this.direction = "left";
    this.draw();
}

public void lookRight() {
    this.erase();
    this.direction = "right";
    this.draw();
}

public void frown() {
    this.erase();
    this.emotion = "frown";
    this.draw();
}

public void smile() {
    this.erase();
    this.emotion = "smile";
    this.draw();
}

public void walk(double dist) {
    this.erase();
    if ( this.direction.equals("right") ) {
        this.figX = this.figX + dist ;
    }
    else {
        this.figX = this.figX - dist ;
    }
}
```

© Peter Andreae

CartoonCharacter: methods

COMP112: 218

```
public void speak(String msg) {
    double bubX = this.figX - ...; // and bubY, bubWd, bubHt
    UI.drawOval(bubX, bubY, bubWd, bubHt);
    UI.drawString(msg, bubX+9, bubY+bubHt/2+3);
    UI.sleep(500);
    UI.eraseRect(bubX, bubY, bubWd, bubHt);
}

public void erase() {
    UI.eraseRect(this.figX, this.figY, this.wd, this.ht);
}

public void draw() {
    String filename = this.imageFolder + "/" + this.direction + "-" +
        this.emotion + ".png" ;
    UI.drawImage(filename, this.figX, this.figY, this.wd, this.ht);
    UI.sleep(500);
}
```

© Peter Andreae

Running the program: main

COMP112: 219

> java CartoonStory or call main on the class from BlueJ

```
public static void main(String[] args){
    CartoonStory cs = new CartoonStory();
}
```

cs: CartoonStory-3

CartoonStory-3

Very simple object!
- no fields
- no constructor

© Peter Andreae

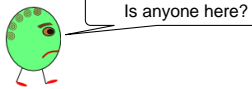
CartoonStory Program: playStory

COMP112: 220

```
public void playStory(){
    this: CartoonStory-3

    CartoonCharacter cf1 = new CartoonCharacter(150, 100, "green");
    cf1.lookLeft();
    cf1.lookRight();
    cf1.frown();
    cf1.speak("Is anyone here?");

    CartoonCharacter cf2 = new CartoonCharacter(300, 100, "blue");
    cf2.speak("Hello");
    cf2.lookLeft();
    cf1.smile();
    cf1.speak("Hi there, I'm Jim");
    cf2.speak("I'm Jan");
}
```



CartoonCharacter-24

figX: 150. wd: 40.
figY: 100. ht: 80.
emotion: "smile"
direction: "right"
imageFolder: "green"

© Peter Andreae


CartoonStory Program: playStory

COMP112: 221

```
public void playStory(){
    this: CartoonStory-3

    CartoonCharacter cf1 = new CartoonCharacter(150, 100, "green");
    cf1.lookLeft();
    cf1.lookRight();
    cf1.frown();
    cf1.speak("Is anyone here?");

    CartoonCharacter cf2 = new CartoonCharacter(300, 100, "blue");
    cf2.speak("Hello");
    cf2.lookLeft();
    cf1.smile();
    cf1.speak("Hi there, I'm Jim");
    cf2.speak("I'm Jan");
}
```



CartoonCharacter-27

figX: 300. wd: 40.
figY: 100. ht: 80.
emotion: "smile"
direction: "right"
imageFolder: "blue"

© Peter Andreae

Keeping track of Multiple objects

COMP112: 222

CartoonCharacter-24

figX: 150. wd: 40.
figY: 100. ht: 80.
emotion: "frown"
direction: "right"
imageFolder: "blue"

CartoonCharacter-27

figX: 300. wd: 40.
figY: 100. ht: 80.
emotion: "smile"
direction: "right"
imageFolder: "blue"

```
cf2.lookLeft();
cf1.smile();

public void lookLeft() {
    this: CartoonCharacter-
    this.erase();
    this.direction = "left";
    this.draw();
}
```

© Peter Andreae

Keeping track of Multiple objects

COMP112: 223

CartoonCharacter-24

figX: 150. wd: 40.
figY: 100. ht: 80.
emotion: "frown"
direction: "right"
imageFolder: "blue"

CartoonCharacter-27

figX: 300. wd: 40.
figY: 100. ht: 80.
emotion: "smile"
direction: "left"
imageFolder: "blue"

```
cf2.lookLeft();
cf1.smile();

public void smile() {
    this: CartoonCharacter-
    this.erase();
    this.emotion = "smile";
    this.draw();
}
```

© Peter Andreae

Menu

- Another example of defining objects
- Scope, Extent, Visibility
- Event-Driven Input

Admin

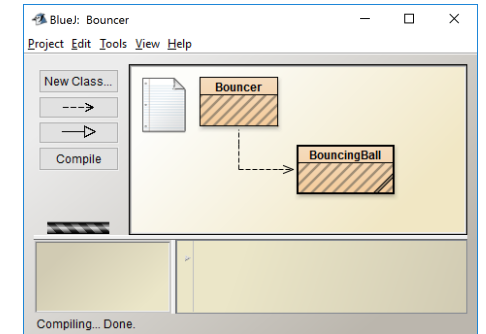
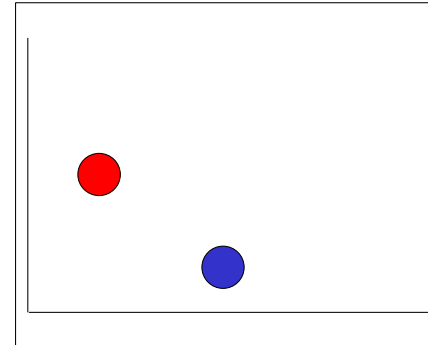
- Test marks
- Beijing Summer School on mobile apps development
 - July 3 -14
 - Two students will be sent
 - See the forum message
 - Email Ian Welch if you are interested

COMP112: 224

© Peter Andreae

Bouncing Balls

- Two classes: Bouncer and BouncingBall



COMP112: 225

© Peter Andreae

Designing Bouncer (“top level” class)

- How does the user interaction work?
 - buttons,
 - constructor
- What are the methods?

COMP112: 226

© Peter Andreae

Designing BouncingBall class

- What fields does it need?
- What methods should it have?
- What should happen when it is first created?

COMP112: 227

© Peter Andreae

BouncingBall: fields & constructor

COMP112: 228

```
public class BouncingBall {  
    // fields  
    private double xPos;  
    private double height;  
    private double xSpeed;  
    private double ySpeed;  
    private Color col;  
  
    // constructor  
    public BouncingBall(double x, double y, double sp){  
    }  
}
```

© Peter Andreae

BouncingBall: methods

COMP112: 229

```
public void draw () {  
  
}  
  
public void move() {  
  
}  
  
public double getX() {  
  
}
```

© Peter Andreae

Places: variables vs fields

COMP112: 230

- Two kinds of places to store information:
- Variables (including parameters)
 - defined inside a method
 - specify places on a worksheet
 - temporary – information is lost when worksheet is finished
 - new place created every time method is called (each worksheet)
 - only accessible from inside the method.
- Fields
 - defined inside a class, but not inside a method
 - specify places in an object
 - long term – information lasts as long as the object
 - new place created for each object
 - accessible from all methods in the class, and from constructor.

© Peter Andreae

Extent and scope

COMP112: 231

- A place with a value must be accessible to some code at some time.
- **Extent:** how long it will be accessible
 - local variables (and parameters) in methods have a limited extent
⇒ only until the end of the current invocation of the method
 - fields have indefinite extent
⇒ as long as the object exists
- **Scope:** what parts of the code can access it
 - Full scope rules are complicated!!!
 - local variables: accessible only to statements
 - inside the block { ... } containing the declaration
 - after the declaration
 - fields: at least visible to the containing class; maybe further.

© Peter Andreae

Scope of variables

COMP112: 232

//read info from file and display

```
while (scan.hasNext()) {  
    String ans = scan.next();  
    if ( ans.equals("flower") ) {  
        Color center = Color.red;  
        int diam = 30;  
    }  
    else if (ans.equals("bud") ) {  
        Color center = Color.green;  
        int diam = 15;  
    }  
    :  
    UI.setColor(center);  
    UI.fillOval(x, y, diam, diam);  
    :  
}
```

different variables!

Out of scope

```
while (scan.hasNext()) {  
    String ans = scan.next();  
    Color center ;  
    int diam ;  
    if ( ans.equals("flower") ) {  
        center = Color.red;  
        diam = 15;  
    }  
    else if (ans.equals("bud") ) {  
        center = Color.blue;  
        diam = 30;  
    }  
    :  
    UI.setColor(center);  
    UI.fillOval(x, y, diam, diam);  
    :  
}
```

may not be initialised

How do you fix it?

© Peter Andreae

Fields: scope, visibility, encapsulation

COMP112: 233

- Fields are accessible to all code in all the (ordinary) methods in the class.
- Should they be accessible to methods in other classes?
 - ⇒ **visibility**: **public** or **private**
 - **public** means that methods in other classes can access the fields
cfg1.figX = 30 in the **CartoonStory** class would be OK
 - **private** means that methods in other classes **cannot** access the fields
cfg1.figX = 30 in the **CartoonStory** class would be an error.

The principle of encapsulation says

- Keep fields private.
- Provide methods to access and modify the fields, if necessary

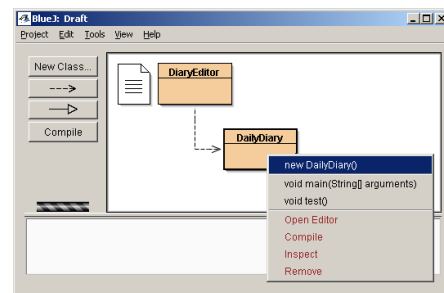
⇒ LDC 5.3

© Peter Andreae

GUI's and Event driven input

COMP112: 237

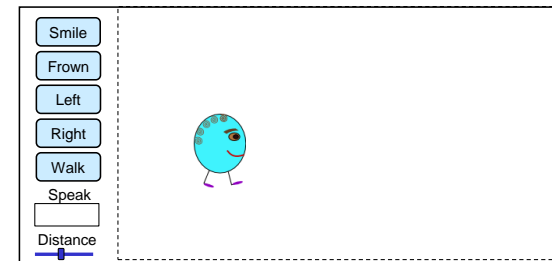
- In a GUI, the interaction is controlled by the user, not by the program
- User initiates "events"
 - buttons
 - menus
 - mouse press/release/drag
 - text fields
 - sliders
 - keys
- Program responds



© Peter Andreae

PuppetMaster

COMP112: 238



- How does Java respond to buttons etc?
 - When a button pressed / text entered in box / slider changed / mouse clicked:
 - Java looks up the object & method attached to the button/box/etc
 - Calls the method
 - passing the value for box or slider.
 - passing kind of action and position (x and y) for mouse.

© Peter Andreae

Setting up event-driven input

COMP112: 239

- Setting up the GUI:
 - To add a button to the UI:
 - specify name of button and method to call
(*object::method* or *class::method*)
(must be a method with no parameters)
 - eg: `UI.addButton("go", this::startGame);`
`UI.addButton("end", UI::quit);`
 - To add a textfield to the UI:
 - Specify name of textfield and method to call
(must be a method with one String parameter)
 - eg `UI.addTextField("name", this::setName);`
 - To add a slider to the UI:
 - Specify name of slider, min, max, initial values, and method to call
(must be a method with one double parameter)
 - eg `UI.addSlider("speed", 10, 50, 20, this::setSpeed);`

© Peter Andreae

Event driven input and fields

COMP112: 240

- Each event will make a new method call.
- \Rightarrow can't remember anything between events in local variables in the methods.
- Typically, need fields in the main object to remember information between events.
 - eg: PuppetMaster has to remember the CartoonCharacter object in a field

© Peter Andreae

PuppetMaster: Design

COMP112: 241

Structure of the PuppetMaster class:

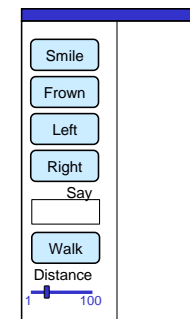
```
public class PuppetMaster ... {  
    // fields to store values between events/method calls  
    private ....  
  
    // Constructor  
    public PuppetMaster(){  
        // set up the buttons, slider, textfield  
        // initialise fields  
    }  
  
    // methods to respond to the buttons, slider, textfield  
    public void ...  
}
```

© Peter Andreae

PuppetMaster: setting up Buttons etc

COMP112: 242

```
public class PuppetMaster ... {  
    // fields  
    // constructor  
    public PuppetMaster(){  
        this.setupGUI();  
    }  
    public void setupGUI(){  
        UI.addButton( "Smile", this::doSmile);  
        UI.addButton( "Frown", this::doFrown);  
        UI.addButton( "Left", this::doLeft);  
        UI.addButton( "Right", this::doRight);  
        UI.addTextField( "Say", this::doSpeak);  
        UI.addButton( "Walk", this::doWalk);  
        UI.addSlider( "Distance", 1, 100, 20, this::setDist);  
        ...  
    }  
    // methods to respond  
}
```



© Peter Andreae

Responding to buttons and textFields

COMP112: 243

```
public class PuppetMaster {
    // fields
    // constructor
    public void setupGUI(){
        UI.addButton("Smile", this::doSmile);
        UI.addButton("Frown", this::doFrown);
        :
        UI.addTextField("Say", this::doSpeak);
    }
    public void doSmile(){
        // tell the CartoonCharacter to smile
    }
    public void doFrown(){
        // tell the CartoonCharacter to frown
    }
    public void doSpeak(String words){
        // tell the CartoonCharacter to say the words
    }
}
```

Methods called by buttons
must have no parameters

Methods called by a textField
must have one String parameter

© Peter Andreae

PuppetMaster: Using Fields

COMP112: 244

Actions on the CartoonCharacter happen in response to different events
⇒ will be in different method calls
⇒ need to store character in a field, not a local variable.

```
public class PuppetMaster{
    // fields
    private CartoonCharacter cc = new CartoonCharacter(200, 100, "blue");
    // constructor
    public void setupGUI(){
        UI.addButton("Smile", this::doSmile); // call doSmile on this
        UI.addButton("Frown", this::doFrown);
        :
    }
    public void doSmile(){
        this.cc.smile();
    }
    public void doFrown(){
        this.cc.frown();
    }
}
```

© Peter Andreae

PuppetMaster: Using Fields

COMP112: 245

Actions on the CartoonCharacter happen in response to different events
⇒ will be in different method calls
⇒ need to store character in a field, not a local variable.

```
public class PuppetMaster{
    // fields
    private CartoonCharacter cc = new CartoonCharacter(200, 100, "blue");
    // constructor
    public void setupGUI(){
        UI.addButton("Smile", cc::smile); // call smile on the cc object, directly
        UI.addButton("Frown", cc::frown);
        :
    }
    public void doSmile(){
        this.cc.smile();
    }
    public void doFrown(){
        this.cc.frown();
    }
}
```

© Peter Andreae

PuppetMaster: TextFields (boxes)

COMP112: 246

```
public class PuppetMaster{
    private CartoonCharacter cc = new CartoonCharacter(200, 100, "blue");
    public PuppetMaster(){
        this.setupGUI();
    }
    public void setupGUI(){
        UI.addButton("Smile", this::doSmile); // call doSmile on this
        UI.addButton("Frown", this::doFrown);
        UI.addTextField("Say", this::doSpeak); // or cc::speak
        :
    }
    public void doSmile(){
        this.cc.smile();
    }
    :
    public void doSpeak(String words){
        this.cc.speak(words);
    }
}
```

© Peter Andreae

PuppetMaster: Sliders

COMP112: 247

```
public class PuppetMaster {
    private CartoonCharacter cc = new CartoonCharacter(200, 100, "blue");
    private double walkDist = 20;
    public PuppetMaster(){
        this.setupGUI();
    }
    public void setupGUI(){
        UI.addButton("Smile", this::doSmile);
        :
        UI.addButton("Walk", this::doWalk);
        UI.addSlider("Distance", 1, 100, 20, this::setDist);
    }
    :
    public void doWalk() {
        this.cc.walk(this.walkDist);
    }
    public void setDist(double value){
        this.walkDist = value;
    }
}
```

Typical design:
field to store value
from one event,
for use by another event

A method called by
a slider must have
one double parameter

© Peter Andreae

GUI: Mouse input

COMP112: 248

- Just like buttons, except don't have to put anything on screen
 - Each press / release / click on the graphics pane will be an event
- Must tell UI object::method to call when a mouse event occurs
UI.addMouseListener(this::doMouse);
- Must define method to say how to respond to the mouse
parameters: kind of mouse event and position of mouse event

```
public void doMouse(String action, double x, double y) {
    if (action.equals("pressed")) {
        // what to do if mouse button is pressed
    }
    else if (action.equals("released")) {
        // what to do if mouse button is released
    }
    else if (action.equals("clicked")) {
        // what to do if mouse button is clicked
    }
}
```

where action
occurred

press-release
in same place

© Peter Andreae

Using the mouse.

COMP112: 249

- Want to let user specify input with the mouse,
 - eg: drawing lines



- Typical pattern:
 - On "pressed",
 - just remember the position
 - On "released",
 - do something with remembered position and new position

© Peter Andreae

Mouse Input

COMP112: 250

/**Let user draw lines on graphics pane with the mouse. */

```
public class LineDrawer {
    private double startX, startY; // fields to remember "pressed" position
    public LineDrawer(){
        UI.setLineWidth(10);
        UI.addMouseListener(this::doMouse);
    }
    public void doMouse(String action, double x, double y) {
        if (action.equals("pressed")) {
            this.startX = x;
            this.startY = y;
        }
        else if (action.equals("released")) {
            UI.drawLine(this.startX, this.startY, x, y);
        }
    }
}
```

© Peter Andreae

Selecting Colors: JColorChooser

COMP112: 251

```
public class LineDrawer {
    private double startX, startY;
    private Color currentColor = Color.black;

    public LineDrawer () {
        UI.addMouseListener(this::doMouse);
        UI.addButton("Color", this::doChooseColour);
    }

    public void doMouse(String action, double x, double y) {
        if (action.equals("pressed") ) { this.startX = x; this.startY = y; }
        else if (action.equals("released") ) { UI.drawLine(this.startX, this.startY, x, y); }
    }

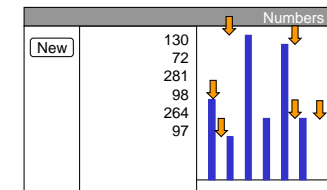
    public void doChooseColour(){
        this.currentColor = JColorChooser.showDialog(null, "Choose Color", this.currentColor);
        UI.setColor(this.currentColor);
    }
}
```

© Peter Andreae

Numbers program

COMP112: 253

- Program for constructing files of numbers:
 - Allow user to select a new file
 - Allow user to enter a set of numbers with the mouse (height of mouse click is the number)
 - Display numbers as bar chart and list in text pane
 - Save numbers to the file as they are entered
- User Interface:
 - Button to clear screen and select new file.
 - Graphics pane to select (with mouse) and display the numbers
 - Text pane to display list of numbers

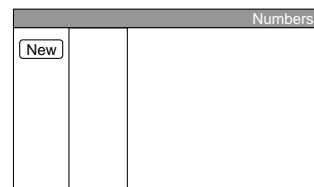


© Peter Andreae

Numbers: Design

COMP112: 254

- Design:
 - When does something happen?
 - button presses
 - mouse clicks
 - Fields
 - to store the file (PrintStream) that the numbers are being saved to
 - to remember the horizontal position of the next bar.
 - Constructor
 - set up the interface
 - Methods to respond to mouse
 - record a new number
 - Method to respond to button
 - clear and start a new file



© Peter Andreae

Numbers: Design

COMP112: 255

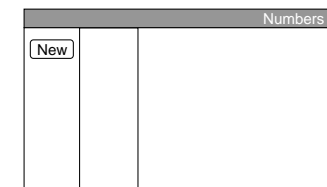
```
public class Numbers {
    private PrintStream outputFile;
    private double barX = 0;
    private static final double BASE= 450;

    public Numbers(){
        UI.addMouseListener(this::doMouse);
        UI.addButton("New", this::doNew);
        UI.drawLine(0, BASE, 600, BASE);
    }

    public void doNew() { ...

    public void doMouse( ...

    public static void main(String[] args){
        new Numbers();
    }
}
```



© Peter Andreae

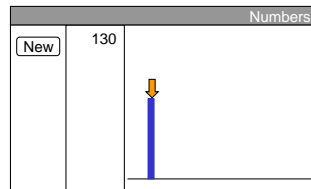
Respond to Mouse:

COMP112: 256

- When user clicks/releases:
 - work out the number they meant
 - draw a bar on the graphics pane
 - display it in the text pane
 - print it to the file

```
public void doMouse(String action, double x, double y) {  
    if (action.equals("released")) {  
        double number = BASE - y;  
        this.barX = this.barX + 10;  
        UI.fillRect(this.barX, y, 5, number);  
        UI.println(number);  
        this.outputFile.println(number);  
    }  
}
```

What's the problem?



© Peter Andreae

Respond to "New" button

COMP112: 257

```
public void doNew(){  
    UI.clearPanels();  
    UI.drawLine(0, BASE, 600, BASE);  
    this.barX = 0;  
    this.outputFile.close();  
    try{  
        this.outputFile = new PrintStream(new File(UIFileChooser.save()));  
    } catch(IOException e) { UI.println("File error: "+e); }  
}
```

Still a problem!

```
if (this.outputFile != null) {  
    this.outputFile.close();  
}
```

// Alternative for the long one line:

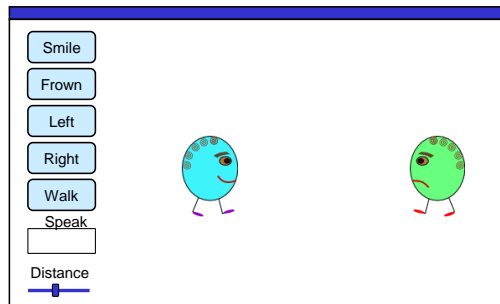
```
String fname = UIFileChooser.save();  
File file = new File(fname);  
this.outputFile = new PrintStream(file);
```

© Peter Andreae

PuppetMaster: Problem 1

COMP112: 258

Suppose we have two characters!



Problem:

- Which character should smile/turn/walk/speak?
- Event-driven input can be tricky!

© Peter Andreae

GUI design: choosing object to act on

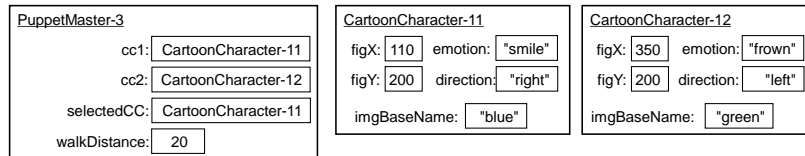
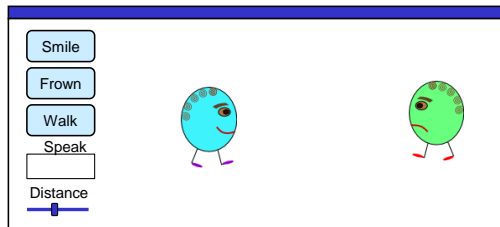
COMP112: 259

- One typical simple GUI interaction mechanism
 - Select object you want to act on
 - Choose action.
- Must remember the currently selected object:
 - in a field, because the action will be performed in a later method
`this.selectedCC = cc1;`
- Typically, the "selected object" doesn't change until user selects another object.

© Peter Andreae

PuppetMaster Problem: two characters

COMP112: 260



© Peter Andreae

PuppetMaster: selecting a character.

COMP112: 261

```
public class PuppetMaster{
    private CartoonCharacter cc1= new CartoonCharacter("blue", 100, 100);
    private CartoonCharacter cc2= new CartoonCharacter("green", 500, 100);
    private CartoonCharacter selectedCC = cc1; // the selected one
    private double walkDistance = 20;

    public PuppetMaster(){
        UI.addButton( "Smile", this::doSmile);
        :
    }

    public void doSmile(){
        this.selectedCC.smile();
    }
    public void doFrown(){
        this.selectedCC.frown();
    }
    :
}
```

How do we change the selected character?

© Peter Andreae

PuppetMaster: buttons for selecting

COMP112: 262

```
public PuppetMaster() {
    UI.addButton( "Jim", this::doJim);
    UI.addButton( "Jan", this::doJan);
    UI.addButton( "Smile", this::doSmile);
    :
}

public void doJim() {
    this.selectedCC = this.cc1;
}

public void doJan() {
    this.selectedCC = this.cc2;
}

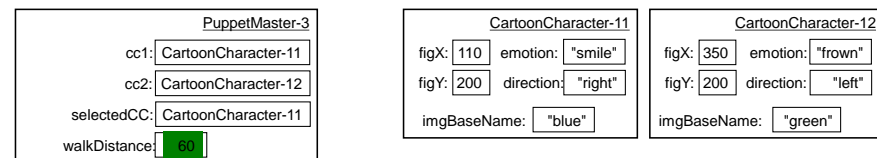
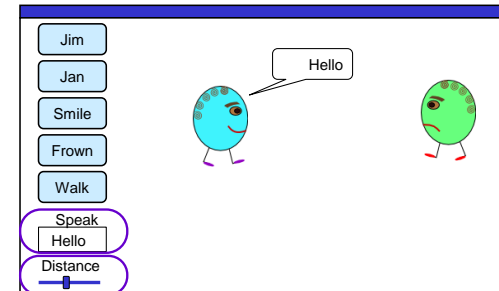
public void doSmile(){
    this.selectedCC.smile();
}

public void doWalk() {
    this.selectedCC.walk(this.walkDistance );
}
```

© Peter Andreae

PuppetMaster: TextFields & Sliders

COMP112: 263



© Peter Andreae

Shorthand: “Lambda expressions”

COMP112: 264

```
public class PuppetMaster{
    private CartoonCharacter selectedCC = new CartoonCharacter(200, 100, "blue");

    public PuppetMaster(){
        UI.addButton("Smile", this::doSmile);
        UI.addButton("Frown", this::doFrown);
        UI.addTextField("Say", this::doSpeak);
        :
    }
    public void doSmile(){
        this.selectedCC.smile();
    }
    public void doFrown(){
        this.selectedCC.frown();
    }
    public void doSpeak(String words){
        this.selectedCC.speak(words);
    }
}
```

Lots of typing for just one line

© Peter Andreae

Shorthand: “Lambda expressions”

COMP112: 265

```
public class PuppetMaster{
    private CartoonCharacter selectedCC = new CartoonCharacter(200, 100, "blue");

    public PuppetMaster(){
        UI.addButton("Smile", () -> { this.selectedCC.smile(); } );
        UI.addButton("Frown", this::doFrown);
        UI.addTextField("Say", this::doSpeak);
        :
    }
    public void doSmile(){
        this.char.smile();
    }
    public void doSmile(){
        this.selectedCC.smile();
    }
    public void doSpeak(String words){
        this.selectedCC.speak(words);
    }
}
```

Lambda Expression:
Unnamed method!!
- has parameters
- has body
- has no name
It is a value!!

© Peter Andreae

Shorthand: “Lambda expressions”

COMP112: 266

```
public class PuppetMaster{
    private CartoonCharacter selectedCC = new CartoonCharacter(200, 100, "blue");

    public PuppetMaster(){
        UI.addButton("Smile", () -> { this.selectedCC.smile(); } );
        UI.addButton("Frown", () -> { this.selectedCC.frown(); } );
        UI.addButton("Left", () -> { this.selectedCC.lookLeft(); } );
        UI.addButton("Right", () -> { this.selectedCC.lookRight(); } );
        UI.addTextField("Say", (String wds) -> { this.selectedCC.speak(wds); } );
        UI.addButton("Walk", () -> { this.selectedCC.walk(this.walkDist); } );
        UI.addSlider("Distance", 1, 100, 20,
            (double val) -> { this.walkDist = val; } );
    }
}
```

You do NOT HAVE TO USE THESE!!
It is always safe to have an explicit, named method.

© Peter Andreae

Shorthand: “Lambda expressions”

COMP112: 267

```
public class PuppetMaster{
    private CartoonCharacter cc1= new CartoonCharacter("blue", 100, 100);
    private CartoonCharacter cc2= new CartoonCharacter("green", 500, 100);
    private CartoonCharacter selectedCC= cc1; // the selected one
    private double walkDistance = 20;

    public PuppetMaster(){
        UI.addButton("Jim", () -> { this.selectedCC = cc1; } );
        UI.addButton("Jan", () -> { this.selectedCC = cc2; } );
        UI.addButton("Smile", () -> { this.selectedCC.smile(); } );
        UI.addButton("Frown", () -> { this.selectedCC.frown(); } );
        UI.addButton("Left", () -> { this.selectedCC.lookLeft(); } );
        UI.addButton("Right", () -> { this.selectedCC.lookRight(); } );
        UI.addTextField("Say", (String wds) -> { this.selectedCC.speak(wds); } );
        UI.addButton("Walk", () -> { this.selectedCC.walk(this.walkDist); } );
        UI.addSlider("Distance", 1, 100, 20,
            (double val) -> { this.walkDist = val; } );
    }
}
```

© Peter Andreae