

Why objects?

- A program has a collection of classes
- Each class has a collection of methods
 - FlagDrawer class had several methods:
 - `public void drawJapanFlag ()`
 - `public void drawFrenchFlag()`
- 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?

?

Classes and Objects

- 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:

String

length, startsWith, substring...

Scanner

next, nextInt, hasNext,...

Butterfly

fly, land ...

PrintStream

println, print, printf...

Animal

goLeft, goRight, jump, speak ...

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

What is an Object

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:

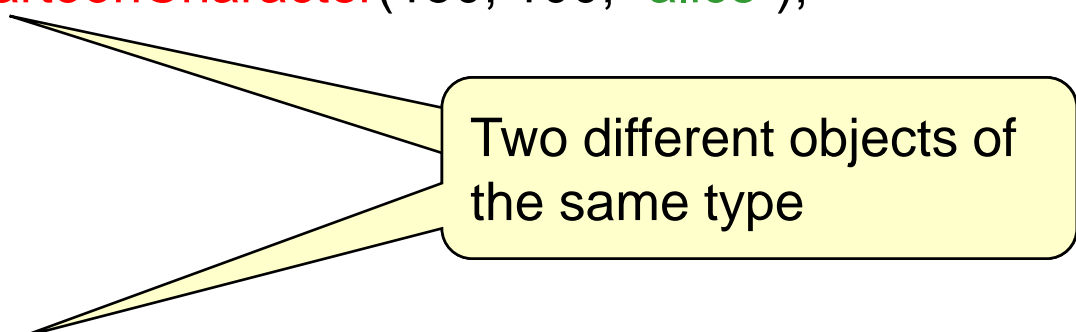
- Fields where data is stored
 - Methods describing the actions
 - Constructor to make new objects
 - Constants
-
- Some objects (top level program objects) may have no data.

CartoonStory program

- Java Program with 2D cartoon objects (similar to the PetShow with Animals)
- 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)

CartoonStory Program

```
public class CartoonStory{
    public void playStory( ){
        CartoonCharacter ca = new CartoonCharacter(150, 100, "alice");
        ca.lookRight();
        ca.lookLeft();
        ca.frown( );
        ca.speak("Is anyone here?");
        CartoonCharacter cb = new CartoonCharacter(300, 100, "bob");
        cb.smile( );  cb.lookLeft( );
        cb.speak("Hello");
        ca.lookRight( );  ca.smile( );
        ca.speak("Hi there, I'm Jim");
        cb.speak("I'm Jan");
    }
}
```



Two different objects of the same type

Defining a class of objects

- 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

CartoonCharacter: methods

```
public class CartoonCharacter {
```

```
    public void lookLeft( ) {  
        // erase figure  
        // change direction  
        // redraw figure  
    }
```

```
    public void frown( ) {  
        // erase figure  
        // change emotion  
        // redraw figure  
    }
```

```
    public void walk(double dist) {  
        // erase figure  
        // change position  
        // redraw figure  
    }
```

```
    public void lookRight( ) {  
        // erase figure  
        // change direction  
        // redraw figure  
    }
```

```
    public void smile( ) {  
        // erase figure  
        // change emotion  
        // redraw figure  
    }
```

```
    public void speak(String msg) {  
        // draw msg in bubble  
        // wait  
        // erase msg  
    }
```

CartoonCharacter: wishful methods

```
public class CartoonCharacter {
```

```
    public void lookLeft( ) {
        this.erase( );
        // change direction
        this.draw( );
    }
```

```
    public void frown( ) {
        this.erase( );
        // change emotion
        this.draw( );
    }
```

```
    public void walk(double dist) {
        this.erase( );
        // change position
        this.draw( );
    }
```

```
    public void erase( ) {
        ???
    }
```

```
    public void lookRight( ) {
        this.erase( );
        // change direction
        this.draw( );
    }
```

```
    public void smile( ) {
        this.erase( );
        // change emotion
        this.draw( );
    }
```

```
    public void speak(String msg) {
        // draw msg in bubble
        // wait
        // erase msg
    }
```

```
    public void draw( ) {
        ???
    }
```


CartoonCharacter: draw

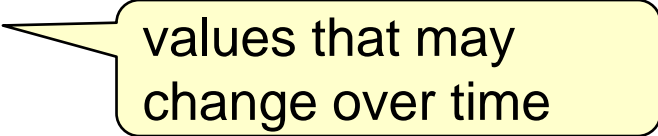
```
public void draw( ) {  
    // work out which image to use (eg, "alice-right-smile.png")  
    // draw the image on the graphics pane  
    // wait a bit  
}
```

```
public void draw( ) {  
    String filename = imagePrefix+"-"+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?

Remembering state

- Each CartoonCharacter object must remember:
 - its state:
 - position
 - emotion
 - direction
 - the folder of image files that it is using.
 - its size
- Can't be stored in local variables in a method
 - local variables are “lost” when the method finishes.
- Have to be stored in the Object itself
 - ⇒ **fields**



values that may change over time

CartoonCharacter Objects

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

CartoonCharacter-24

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

CartoonCharacter-27

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

- Objects are like entries in your Contacts

Using fields:

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

`this . fieldname`

eg:

note: fields have no ()

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

Object the method
was called on

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

Using fields:

Object

CartoonCharacter-24

figX:	150	wd:	40
figY:	300	ht:	80
emotion:	"smile"	imagePrefix:	"alice"
direction:	"right"		

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

cf1: CartoonCharacter-24

ID of Object

Method worksheet

```
public void lookLeft( ) {
```

this: CartoonCharacter-

```
    this.erase( );
    this.direction = "left";
    this.draw( );
```

```
}
```



Using fields:

Object

CartoonCharacter-24

figX:	150	wd:	40
figY:	300	ht:	80
emotion:	"smile"	imagePrefix:	"alice"
direction:	"left"		

Method Worksheet

```
public void draw( ) {
```

```
    this: CartoonCharacter-
```

```
        String filename = this.imagePrefix + "-" + this.direction + "-" +
            this.emotion + ".png" ;
```

```
        UI.drawImage(filename, this.figX, this.figY, this.wd, this.ht);
        UI.sleep(500);
```

```
    }
```



Using fields:

Object

CartoonCharacter-24

figX:	150	wd:	40
figY:	300	ht:	80
emotion:	"smile"	imagePrefix:	"alice"
direction:	"left"		

```

:
✓ cfg1.lookLeft( );
  cfg1.walk(20);
:

```

cfg1: CartoonCharacter-24

```

public void lookLeft( ) {
    this: CartoonCharacter-24

```

```

    ✓ this.erase( );
    ✓ this.direction = "left";
    ✓ this.draw( );
}

```



Using fields:

Object

CartoonCharacter-24

figX:	150	wd:	40
figY:	300	ht:	80
emotion:	"smile"	imagePrefix:	"alice"
direction:	"left"		

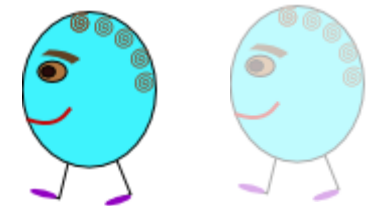
✓ `cfg1.lookLeft();`
`cfg1.walk(20);`
`:`

cfg1: `CartoonCharacter-24`

```
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( );
}
```





Objects and Classes

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

Setting up an object

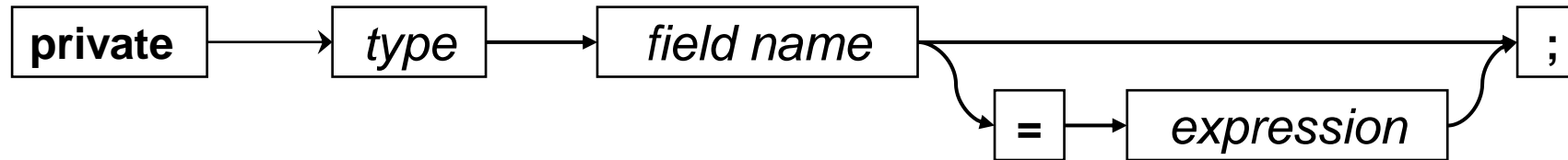
Must declare the Fields of an object

- 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 like local variables
must be declared

Syntax of Field declarations:



```
public class CartoonCharacter {
```

```
// fields
```

```
private double figX;
```

```
private double figY;
```

```
private String direction = "right";
```

```
private String emotion = "smile";
```

```
private String imagePrefix;
```

```
private double wd = 40;
```

```
private double ht=80;
```

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

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

```
// current position of character
```

```
// current direction it is facing
```

```
// current emotion
```

```
// base name of images
```

```
// dimensions of figure
```

Every CartoonCharacter will start with these values in the fields.

Setting up an object

- 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, "alice")`)

CartoonCharacter class

```
public class CartoonCharacter {
```

```
// fields
```

```
private double figX, figY;
```

```
private String direction = "right";
```

```
private String emotion = "smile";
```

```
private String imagePrefix;
```

```
private double wd = 40, ht=80;
```

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

```
// current position of figure
```

```
// current direction it is facing
```

```
// current emotion
```

```
// folder where images stored
```

```
// dimensions
```

```
// constructor
```

```
public CartoonCharacter(double x, double y, String prefix){
```

```
    this.imagePrefix = prefix;
```

```
    this.figX = x;
```

```
    this.figY = y;
```

```
    this.draw();
```

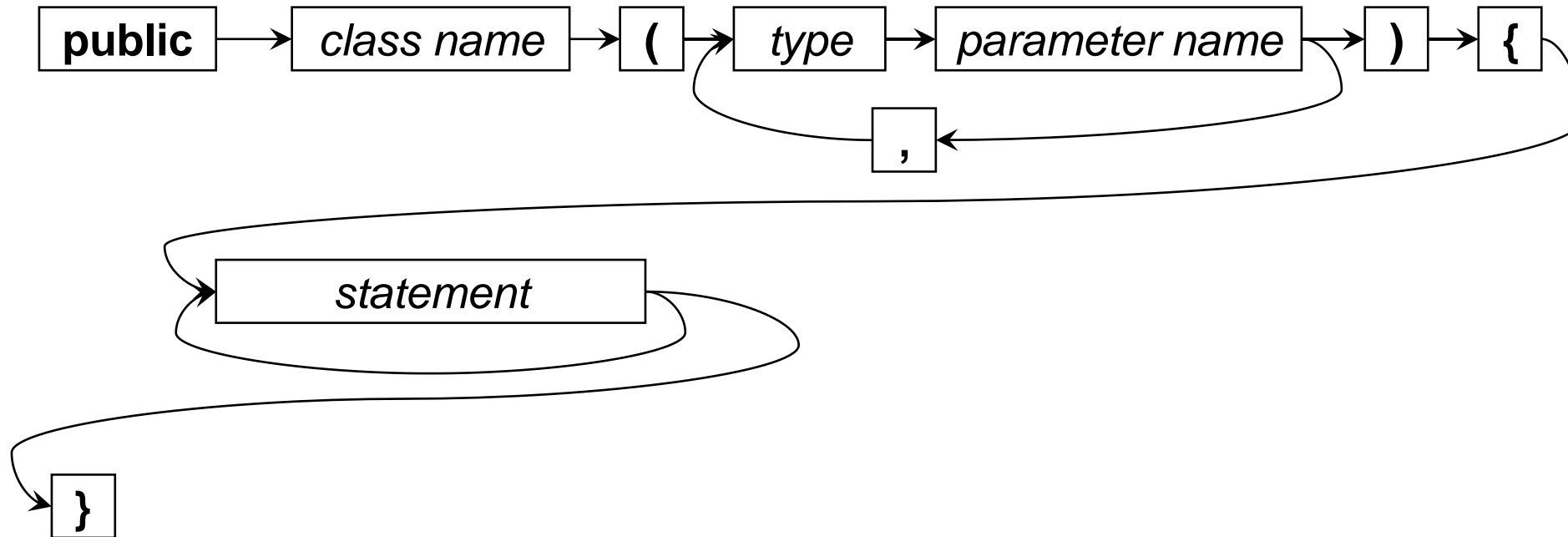
```
}
```

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

```
public void lookLeft() {
```

```
    this.erase(); .....
```

Syntax of Constructor Definitions



```

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

Constructors

- 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.

What happens with **new** ?

When an object is created

eg `new CartoonCharacter(100, 200 , "bob");`

- 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 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:

figY:

emotion:

direction:

imagePrefix:

wd:

ht:



The whole Program

```
public class CartoonStory{  
    public void playStory(){  
        CartoonCharacter cf1 = new CartoonCharacter(150, 100, "alice");  
        cf1.lookLeft();  
        cf1.lookRight();  
        cf1.frown();  
        cf1.speak("Is anyone here?");  
        CartoonCharacter cf2 = new CartoonCharacter(300, 100, "bob");  
        cf2.speak("Hello");  
        cf2.lookLeft() ;  
        cf1.smile();  
        cf1.speak("Hi there, I'm Jim");  
        cf2.speak("I'm Jan");  
    }  
    public void setupGUI(){  
        UI.addButton("story", this::playStory);  
    }  
    public static void main(String[ ] args){  
        new CartoonStory().setupGUI();  
    }  
}
```

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

CartoonCharacter: fields & constructor

```
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 imagePrefix;    // base name of image set
    private double wd = 40;        // dimensions
    private double ht=80;

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

CartoonCharacter: methods

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

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

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

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

CartoonCharacter: methods

```
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.imagePrefix + "-" + this.direction + "-" + this.emotion + ".png";
    UI.drawImage(filename, this.figX, this.figY, this.wd, this.ht);
    UI.sleep(500);
}
```

CartoonStory Program: playStory

```
public void playStory(){
```

```
this: CartoonStory-3
```

```
    CartoonCharacter cf1 = new CartoonCharacter(150, 100, "alice");
```

```
    cf1.lookLeft();
```

```
    cf1: CartoonCharacter-
```

```
    cf1.lookRight();
```

```
    cf1.frown()
```

```
    cf1.speak("Is anyone here?");
```



```
    CartoonCharacter cf2 = new CartoonCharacter(300, 100, "bob");
```

```
    cf2.speak("Hello");
```

```
    cf2.lookLeft();
```

```
    cf2: CartoonCharacter-
```

```
    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"
```

```
imagePrefix: "alice"
```



Is anyone here?

CartoonStory Program: playStory

```
public void playStory(){
```

```
this: CartoonStory-3
```

```
    CartoonCharacter cf1 = new CartoonCharacter(150, 100, "alice");
```

```
    cf1.lookLeft();
```

```
    cf1.lookRight();
```

```
    cf1.frown();
```

```
    cf1.speak("Is anyone here?");
```

```
cf1: CartoonCharacter-24
```

```
cf2: CartoonCharacter-27
```

```
    ⇒ CartoonCharacter cf2 = new CartoonCharacter(300, 100, "bob");
```

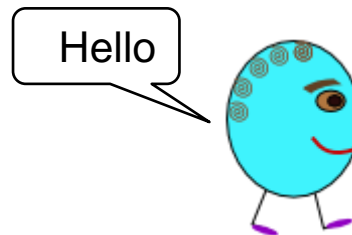
```
    cf2.speak("Hello");
```

```
    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"
```

```
imagePrefix : "bob"
```

Keeping track of Multiple objects

CartoonCharacter-24

figX: 150. wd: 40.

figY: 100. ht: 80.

emotion: "frown"

direction: "right"

imagePrefix : "alice"

CartoonCharacter-27

figX: 300. wd: 40.

figY: 100. ht: 80.

emotion: "smile"

direction: "right"

imagePrefix : "blue"

⇒ `cf2.lookLeft();`
`cf1.smile();`

cf1: CartoonCharacter-24 cf2: CartoonCharacter-27

```
public void lookLeft() {
```

this: CartoonCharacter-27

```
    this.erase();
    this.direction = "left";
    this.draw();
```

```
}
```



Keeping track of Multiple objects

CartoonCharacter-24

figX: 150. wd: 40.

figY: 100. ht: 80.

emotion: "frown"

direction: "right"

imagePrefix : "alice"

CartoonCharacter-27

figX: 300. wd: 40.

figY: 100. ht: 80.

emotion: "smile"

direction: "left"

imagePrefix : "blue"

⋮
 ✓ cf2.lookLeft();
 cf1.smile();

cf1: CartoonCharacter-24 cf2: CartoonCharacter-27

⋮
public void smile() {

this: CartoonCharacter-24

```

this.erase();
this.emotion = "smile";
this.draw();

```

}

