Object oriented programming

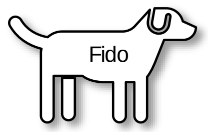
- Key idea of OO programming
  - program structured into classes of objects.
  - each class specifies a kind of object – e.g., the actions it can perform.

- Calling methods in OO languages like java
  - tell an *object* to perform a *method*, passing *arguments*

Fido: run fast!

Fido: get the ball!

Fido: eat!



- Making objects
  - Some objects are predefined.
  - Create objects with bluej:
    - Right-click on class, and select new ……
    - This is how we run programs with BlueJ.
    - not standard, and not a general solution

Objects

Question:

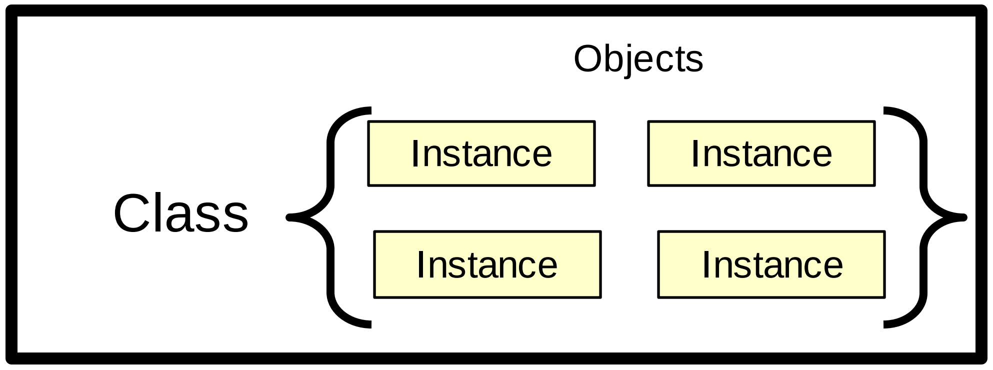
How can a program make new objects?

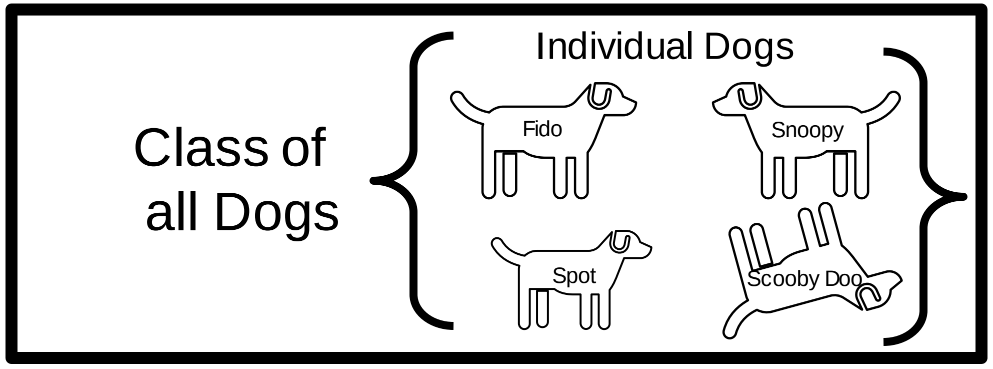
More Questions:

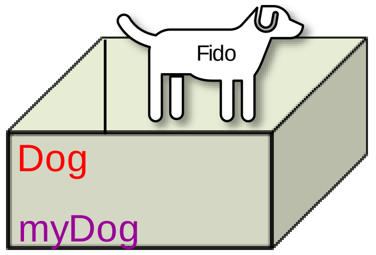
What is an object anyway?

Why do we need them?

- An object is typically a collection of data with a set of actions it can perform.







- The objects we have made so far are a bit strange – no data; just actions. (TemperatureConverter, Drawer)

Examples of objects

Butterfly program

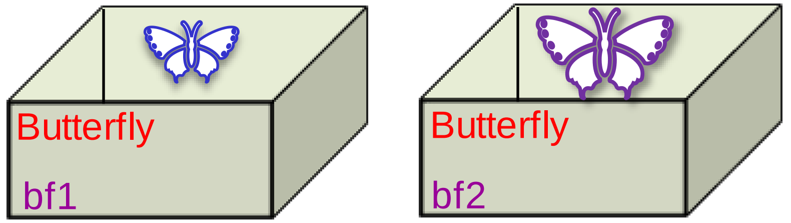
- Each butterfly is represented by an object which stores the state of the butterfly (position, wing state, direction)
- Butterflies have methods
  - move(double dist) and
  - land()

- CartoonFigure program
  - Each cartoon figure is represented by an object which stores the state of the cartoon figure (image, position, direction facing, smile/frown).
  - CartoonFigure objects have methods
    - walk(double dist)
    - smile() frown()
    - lookLeft() lookRight()
    - speak(String words) think(String words)

Using objects

- If the variable bf1 and bf2 contained Butterfly objects, you could do:

```
public void showButterflies(){
    Butterfly bf1 = ?????
    Butterfly bf2 = ?????
    bf1.move(10);
    bf2.move(20);
    bf1.land();
    bf2.move(20);
    bf1.move(5);
}
```



Nothing new here:
Just standard method calls!

Problem:
How do you get a Butterfly object into the variables?

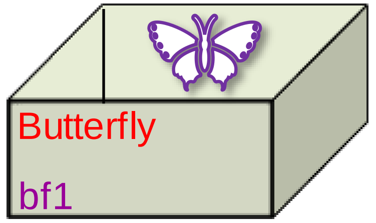
Creating Objects

- Need to construct new objects:
- New kind of expression: **new**

Calling the constructor

```
Butterfly bf1 = new Butterfly(100, 300 )
```

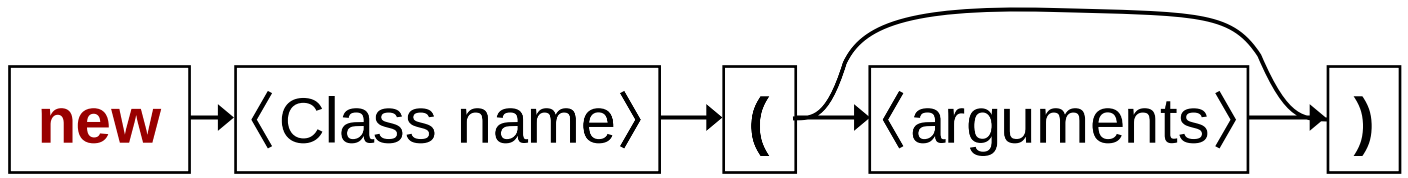
Creates a new object, which is put into bf1



- Constructor calls are like method calls that return a value.
  - have ( )
  - may need to pass arguments
  - returns a value – the new object that was constructed.
- Constructor calls are NOT method calls
  - there is no object to call a method on.
  - must have the keyword **new**
  - name must be the name of the class

Creating Objects: new

```
Butterfly b1 = new Butterfly(100, 300);

UI.setColor( new Color(255, 190, 0) );
```



- Calling a constructor:
  - **new** (a keyword)
  - Butterfly (the type of object to construct)
  - **(…)** (arguments: specifying information needed to construct the new object)
- This is an expression: it returns the new object
  - can put in a variable
  - can use in an enclosing expression or method call

Reading Documentation

- Documentation of a class:
  - Specifies the methods:
    - name
    - type of the return value (or void if no value returned)
    - number and types of the parameters.

```
void    move (double  dist)
        moves the butterfly by dist, in its current direction.
```

  - Specifies the constructors:
    - number and types of the parameters
      (name is always the name of the class,
       return type is always the class)

```
Butterfly(double x, double y)
requires the initial position of the butterfly
```

Bluej lets you see the documentation of your classes

Example: Butterfly Grove program

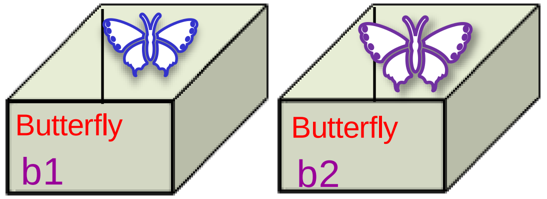
```
public class ButterflyGrove{
  /** A grove of Butterflies which
      fly around and land   */

  public void oneButterfly(){
     Butterfly b1 = new Butterfly(50, 20);
     b1.move(5);
     b1.move(10);
     b1.move(15);
     b1.move(10);
     b1.move(11);
     b1.move(12);
     b1.move(13);
     b1.move(14);
     b1.move(15);
     b1.move(16);
     b1.move(10);
     b1.land();
  }
```

```
  public void twoButterflies(){
     Butterfly b1 =   new Butterfly(100, 20);
     b1.move(5);
     b1.move(10);
     b1.move(15);
     double x = 400*Math.random();
     Butterfly b2 = new Butterfly(x, 40);
     b2.move(10);
     b1.move(15);
     b2.move(10);
     b1.move(12);
     b2.move(10);
     b1.move(11);
     b1.move(7);
     b1.land();
     b2.move(20);
     b2.move(25);
     b2.land();
  }
```

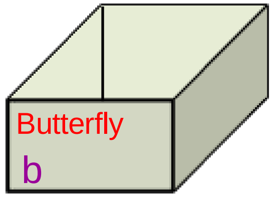
Objects are values too:

- Objects can be passed to methods, just like other values.

```
public void Butterflies(){
    Butterfly b1 =   new Butterfly(100, 20);
    Butterfly b2 = new Butterfly(x, 40);
    this.upAndDown(b1);
    this.upAndDown(b2);
}
```



```
public void upAndDown(Butterfly b){
    b.move(10);
    b.move(15);
    b.land();
    b.move(15);
    b.move(20);
    b.land();
}
```



Designing with methods that call other methods

- Design a Java program to measure reaction time of users responding to true and false "facts".
  - Ask the user about a fact: "Is it true that the BE is a 4 Year degree?"
  - Measure the time they took
  - Print out how much time.

- Need a class
  - what name?
- Need a method
  - what name?
  - what parameters?
  - what actions?

ReactionTimeMeasurer

```
/** Measures reaction times for responding to true-false statements */
public class ReactionTimeMeasurer {

/** Measure and report the time taken to react to a question */
    public void measureReactionTime() {
            ;   // find out the current time and remember it
                // ask the question and wait for answer
            ;   // find out (and remember) the current time
                // print the difference between the two times
    }
}
```

Write the method body in comments first,

(to plan the method without worrying about syntax)

Work out what information needs to be stored (ie, variables)

ReactionTimeMeasurer

```
    /** Measure and report the time taken to react to a question */
    public void measureReactionTime() {
        long startTime = System.currentTimeMillis();
        UI.askString("Is it true that the sky is blue?");
        long endTime = System.currentTimeMillis();
        UI.printf("Reaction time = %d milliseconds \n",  (endTime - startTime) );
    }
}
```

Returns a very big integer
⇒ long
(milliseconds since 1/1/1970

Just asking one question is not enough for an experiment.

➔ need to ask a sequence of questions.

Multiple questions, the bad way

```
/** Measure and report the time taken to react to a question */
public void measureReactionTime(){
    long startTime = System.currentTimeMillis();
    UI.askString( "Is it true that John Quay was the Prime Minister");
    long endTime = System.currentTimeMillis();
    UI.printf("You took %d milliseconds \n",  (endTime - startTime) );

    startTime = System.currentTimeMillis();
    UI.askString( "Is it true that 6 x 4 = 23");
    endTime = System.currentTimeMillis();
    UI.printf("You took %d milliseconds \n",  (endTime - startTime) );

    startTime = System.currentTimeMillis();
    UI.askString( "Is it true that summer is warmer than winter");
    endTime = System.currentTimeMillis();
    UI.printf("You took %d milliseconds \n",  (endTime - startTime) );

    startTime = System.currentTimeMillis();
    UI.askString( "Is it true that Wellington’s population > 1,000,000");
    endTime = System.currentTimeMillis();
    UI.printf("You took %d milliseconds \n",  (endTime - startTime) );
}
```

Lots of repetition.
But not exact repetition.
How can we improve it?

Good design with methods

- Key design principle:
  - Wrap up repeated sections of code into a separate method,
  - Call the method several times:

```
public void measureReactionTime ( ) {
    this.measureQuestion("John Quay was the Prime Minister");
    this.measureQuestion("6 x 4 = 23");
    this.measureQuestion("Summer is warmer than winter");
    this.measureQuestion( "Wellington's population > 1,000,000 ");
}
public void measureQuestion (String fact       ) {
    long startTime = System.currentTimeMillis();
    UI.askString("Is it true that " + fact . );
    long endTime = System.currentTimeMillis();
    UI.printf("You took %d milliseconds \n",  (endTime - startTime) );
}
```

We need to *parameterise* the method

Improving ReactionTimeMeasurer (1)

```
public void measureReactionTime() {
    this.measureQuestion("John Quay was the Prime Minister");
    this.measureQuestion("6  x 4 = 23");
    this.measureQuestion("Summer is warmer than Winter");
    this.measureQuestion("Wellington's population > 1,000,000 ");
}
public void measureQuestion(String fact) {
    long startTime = System.currentTimeMillis();
    UI.askString("Is it true that" + fact);
    long endTime = System.currentTimeMillis();
    UI.printf("You took %d milliseconds \n",  (endTime - startTime) );
}
```

Understanding ReactionTimeMeasurer

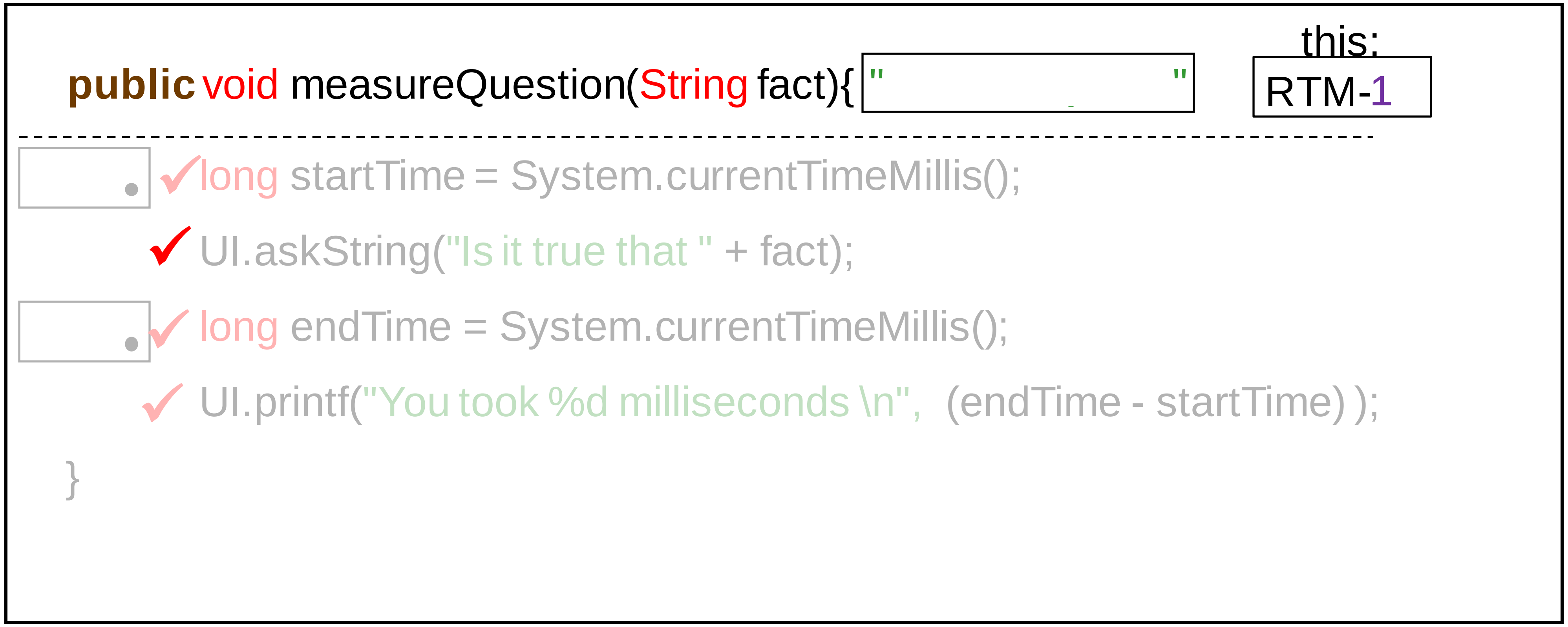
- What happens if we call the method on the object RTM1:

  RTM1 . measureTime();

```
public void measureReactionTime(){                          this:
                                                            RTM-1
-----------------------------------------------------------------------
    this.measureQuestion("John Quay was the Prime Minister");
    this.measureQuestion("6 x 4 = 23");
    this.measureQuestion(“summer is warmer than Winter");
    this.measureQuestion("Wellington’s population >1,000,000");
```

The object the method was called on is copied to "this" place

Understanding method calls



```
public void measureQuestion(String fact){
-------------------------------------------------------------------------------------------
✓ long startTime = System.currentTimeMillis();
✓ UI.askString("Is it true that " + fact);
✓ long endTime = System.currentTimeMillis();
✓ UI.printf("You took %d milliseconds \n",  (endTime - startTime) );
}
```

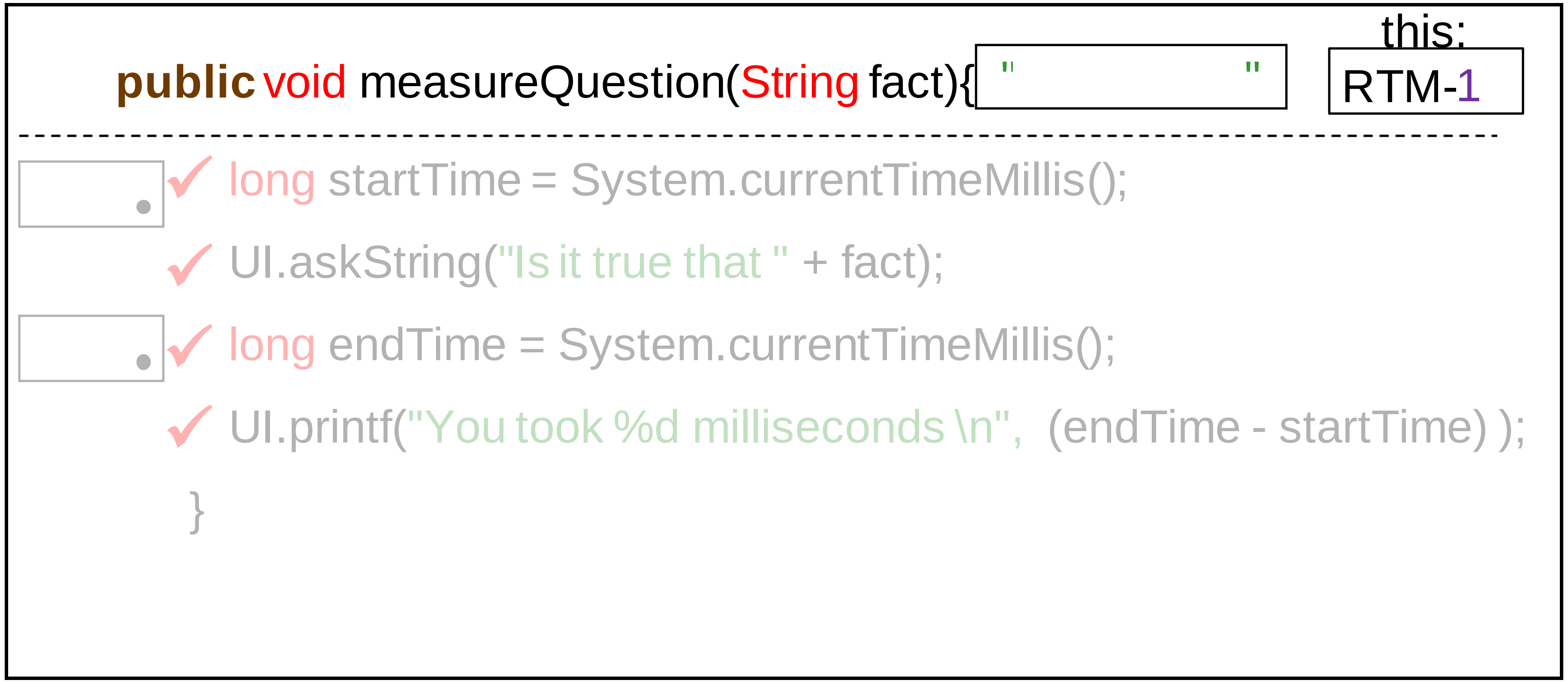
Understanding ReactionTimeMeasurer

this: RTM-1

```
public void measureReactionTime(){
------------------------------------------------------------------------
✓ this.measureQuestion("John Quay was the Prime Minister");
  this.measureQuestion("6 x 4 = 23");
  this.measureQuestion(“summer is warmer than Winter");
  this.measureQuestion("Wellington’s population > 1,000,000");
```

Understanding ReactionTimeMeasurer

New measureQuestion worksheet:



```
public void measureQuestion(String fact){ "            "    this: RTM-1
------------------------------------------------------------------------
✓ long startTime = System.currentTimeMillis();
✓ UI.askString("Is it true that " + fact);
✓ long endTime = System.currentTimeMillis();
✓ UI.printf("You took %d milliseconds \n",  (endTime - startTime) );
 }
```

Each time you call a method,
it makes a fresh copy of the worksheet!

Understanding ReactionTimeMeasurer

this: RTM-1

```
public void MeasureReactionTime(){
------------------------------------------------------------
✓     this. measureQuestion("John Quay was the Prime Minister");
✓     this. measureQuestion("6 x 4 = 23");
      this. measureQuestion(“summer is warmer than Winter");
      this. measureQuestion(" Wellington’s population > 1,000,000");
```

ReactionTimeMeasurer Problem

- A good experiment would measure the average time over a series of trials
  - Our program measures and reports for each trial.

- Need to add up all the times, and compute average:
  - problem:
    - measureReactionTime needs to add up the times
    - measureQuestion actually measures the time, but prints it out.
    - How do we get the time back from measureQuestion to measureReactionTime?

  - We need to make measureQuestion return the time value to measureReactionTime.

Methods that return values

- Some methods just have "effects":

```
UI.println("Hello there!");
UI.printf("%4.2f miles is the same as %4.2f km\n", mile, km);
UI.fillRect(100, 100, wd, ht);
UI.sleep(1000);
```

- Some methods just return a value:

```
long  now = System.currentTimeMillis();
double distance = 20 * Math.random();
double ans = Math.pow(3.5, 17.3);
```

- Some methods do both:

```
double height = UI.askDouble("How tall are you");
Color col =JColorChooser.showDialog(UI.getFrame(),  "paintbrush",  Color.red);
```

Defining methods to return values

Improving ReactionTimeMeasurer:

make measureQuestion **return** a value instead of just printing it out.

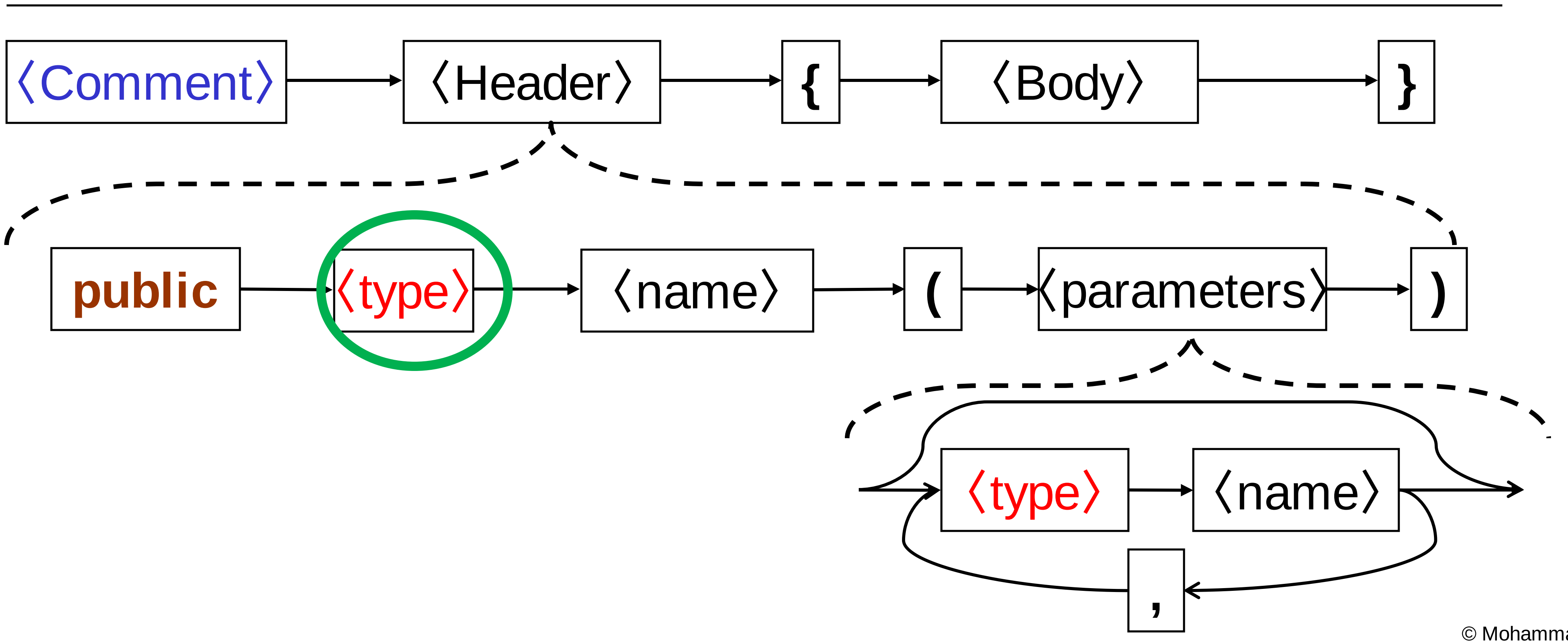
```
public void measureReactionTime() {
    long time = 0;
    time = time + this.measureQuestion("John Quay was the Prime Minister");
    time = time + this.measureQuestion("11 x 13 = 143");
    time = time + this.measureQuestion("Summer is warmer than Winter");
    time = time + this.measureQuestion(" Wellington's pop > 1,000,000 ");
    UI.printf("Average reaction time = %d milliseconds\n", (time / 4));
}
```

Specifies the type of value returned.
void means "no value returned"

```
public long measureQuestion(String fact) {
    long startTime = System.currentTimeMillis();
    ……
}
```

Syntax: Method Definitions (v3: return type)

```
/** Measure time taken to answer a question*/
        public long measureQuestion ( String fact ){
            long startTime = System.currentTimeMillis();
                :
```



Defining methods to return values

If you declare that a method returns a value,
then the method body must return one!

```
public long  measureQuestion(String fact) {
    long startTime = System.currentTimeMillis();
    String  ans = UI.askString("Is it true that " + fact);
    long endTime = System.currentTimeMillis();
    return  (endTime - startTime) ;
}
```

New kind of statement
  Means: exit the method and return the value
  The value must be of the right type

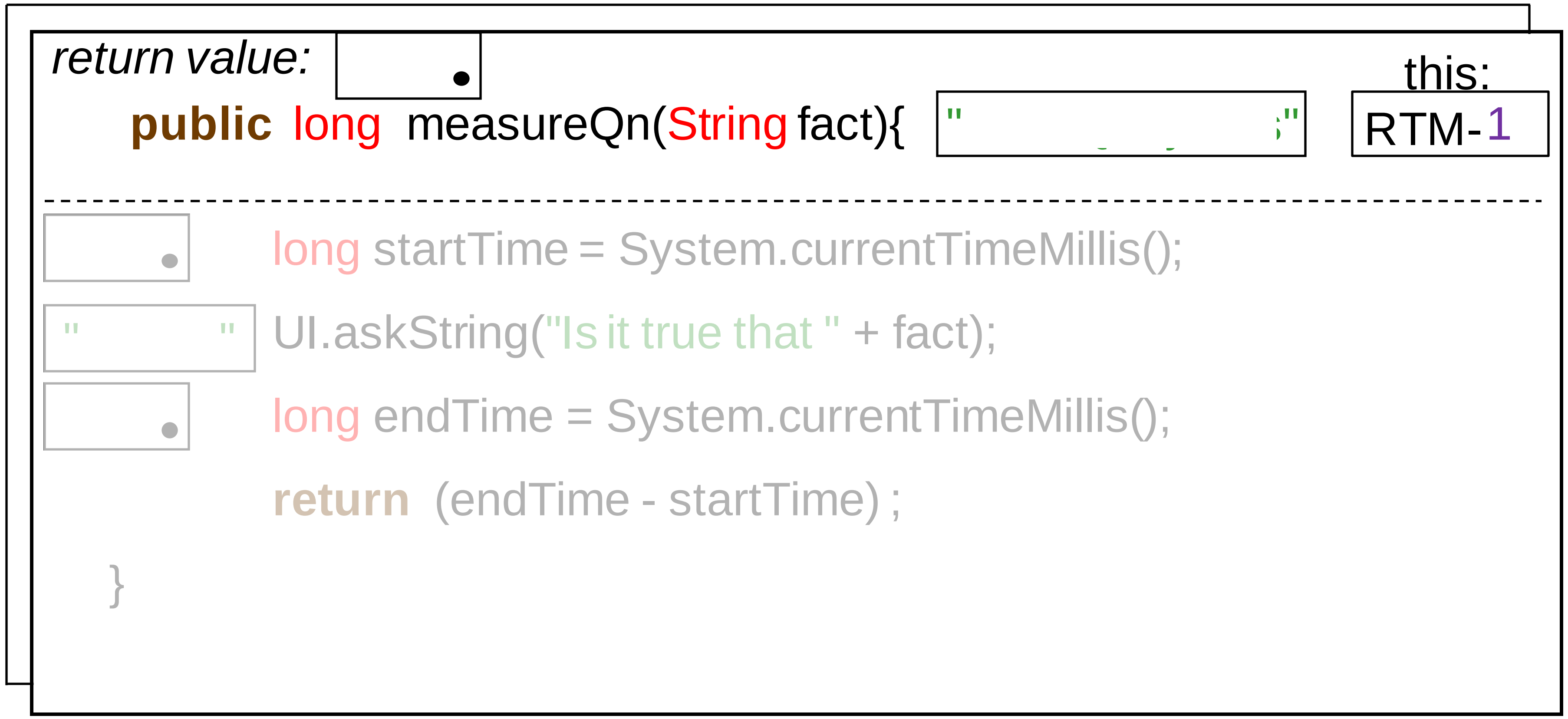
Returning values.

- What happens if we call the method:

  RTM-1 . measureReactionTime();

```
                                                  this:
public void measureReactionTime(){                RTM-1
------------------------------------------------------------
✓ long time = 0;              0 ●

  time = time + this.measureQuestion("John Quay was the Prime Minister");

  time = time + this.measureQuestion("6 x 4 = 23");

  time = time + this.measureQuestion("summer is warmer than Winter");

  time = time + this.measureQuestion("Wellington's pop > 1,000,000");
```

Returning values



```
    public long  measureQn(String fact){
-----------------------------------------------------------------------------
            long startTime = System.currentTimeMillis();
            UI.askString("Is it true that " + fact);
            long endTime = System.currentTimeMillis();
            return  (endTime - startTime) ;
    }
```

Returning values.

- What happens if we call the method:

  RTM-1 . askQuestions();

```
                                                         this:
public void measureReactionTime(){                       RTM-1
-------------------------------------------------------------------
✓ long time = 0;          0.

✓ time = time + this.measureQuestion("John Quay was the Prime Minister");
  time = time + this.measureQuestion("6 x 4 = 23");
  time = time + this.measureQuestion("summer is warmer than Winter");
  time = time + this.measureQuestion(" Wellington's pop > 1,000,000");
```

More about Return

- If a method has a return type, it must have a **return** statement that returns a value

- It must return a value for every possible path

  ⇒ may need several **return** statements:

```
public String fullDayName(String str){
    str = str.toLowerCase();
    if (str.startsWith("m")){
        return "Monday";
    }
    else if (str.startsWith("tu")){
        return "Tuesday";
    }
    else if (str.startsWith("w")){
        return "Wednesday";
    }….
```

More about Return

- **return** does two things:
  - specifies the value that will be returned to the calling method
  - exits the current method, skipping over all remaining statements.

- Methods with a void return type:
  - Can't return a value
  - Can still have a **return** statement (**return**;) with no value.

    ⇒ exit method at this point.

```
public void drawLollipop(double x, double y, double size, double length,){
    if (size < 2 || length < size/2){     // invalid parameters
            return;
    }
    // draw the lollipop
    UI.setColor(Color.red);
    UI.fillRect(x-size/2; y-size/2, size, size);
      :
```

Aside: Random numbers

- Math.random() computes and returns a random double
  - between 0.0 and 1.0
- To get a random number between min and max:
  - min + random number * (max-min)

    (50.0 + Math.random() * 70.0)

    gives a value between 50.0 and 120.0
- This is an expression:
  - can assign it to a variable to remember it
  - can use it inside a larger expression
  - can pass it directly to a method