Solving Problems with Computers

Slides for the four videos introducing problem solving using computers

Algorithms

- What is an algorithm and how does it relate to programming?

Using ‘primitive’ data

- The basic data types available for modelling ‘things’ in our programs

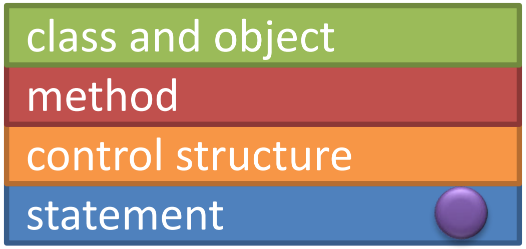
Importing & using objects

- Using larger pieces of code written by others

Some commonly used objects

- …that you will use in this unit and beyond

Solving Problems with Computers: Algorithms

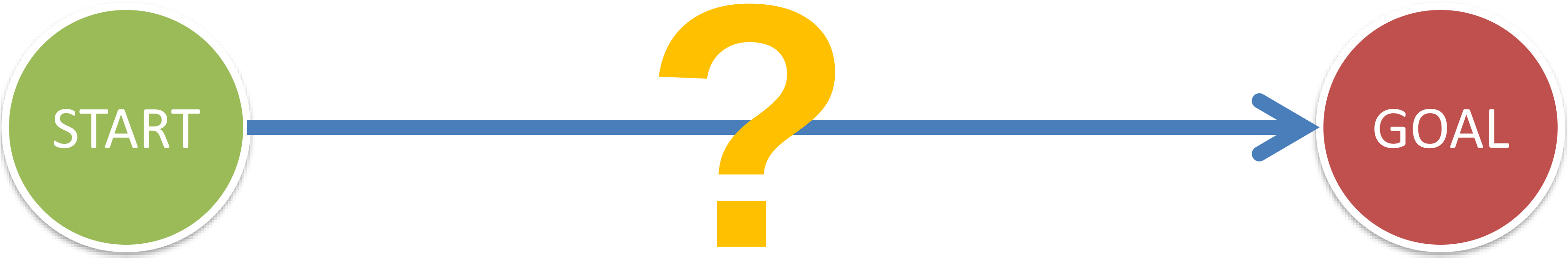


03 Problem Solving with Computers

cc BY NC SA

Dr James Montgomery, james.montgomery@utas.edu.au

What is a problem?



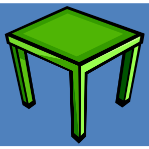
- known input; or
- physical location; or
- existing application; or
- …

- calculated result; or
- different location; or
- new function; or
- …

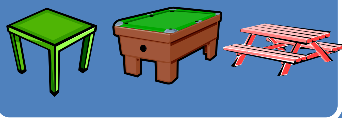
Problem solving in general

Understand the problem

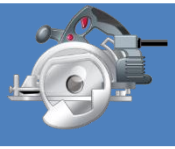
Dissect the problem into manageable pieces



Design a solution



Consider alternatives to the solution and refine it



Implement the solution

Test the solution and fix any problems that exist

Algorithms

An **algorithm** is a set of instructions for performing a task:

	Sequence	order is important; must be able to determine the next step
of	Steps	each instruction must be clear and executable
that eventually	Stops	the process must terminate
and which	Succeeds	the goal must be achieved on termination

Algorithms

Dish:

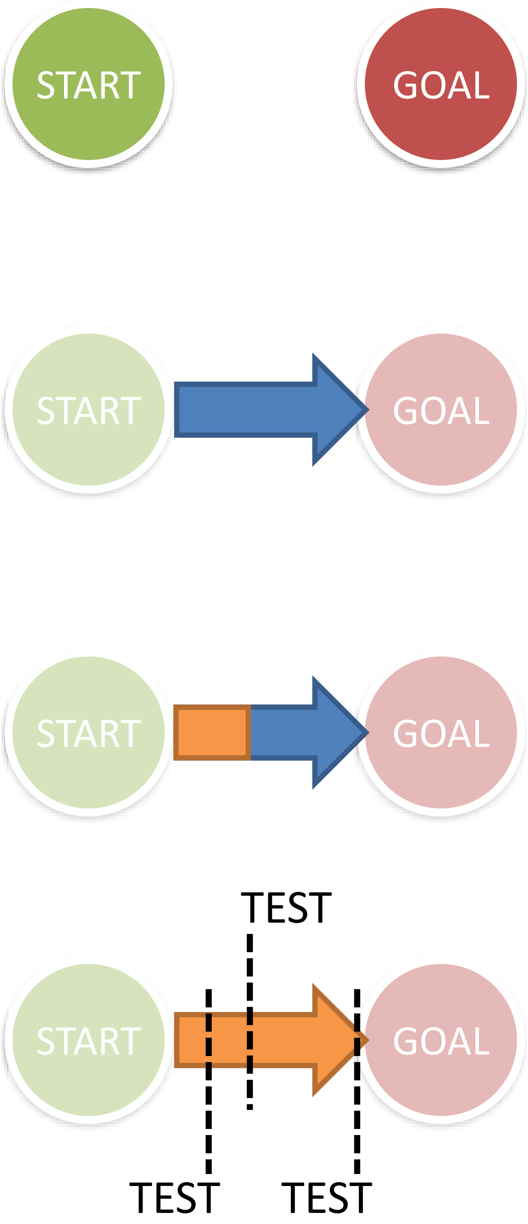
Recipe

Serves:

WikimediaCommons/WillowW

```
public class XIIX extends Sonnet {
  public static void main(String[] args) {
    System.out.println("Shall I compare thee...
    System.out.println("Thou art more lovely...
    System.out.println("Rough winds to shake...
    System.out.println("And summer's lease...
    System.out.println("Sometime too hot the...
    System.out.println("And often is his gold...
  }
}
```

Program Development



Establish requirements

WHAT the program must do

Create design; how to solve the problem

The ALGORITHM

Implement the code

Use INCREMENTAL development

Test the implementation (incrementally)

The longer an error stays in a program the more expensive it is to fix

‘High Level’ algorithms

Use existing *libraries* of code

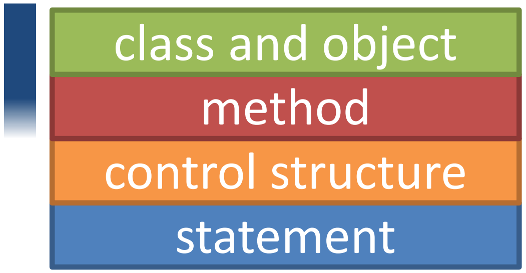
- organised into classes of objects

Create objects that will be used to solve the problem

- Interact with them via their *methods*

Documentation is important

- What *classes* of *objects* exist?
- What *methods* do they have?
- How do the methods work?



class and object

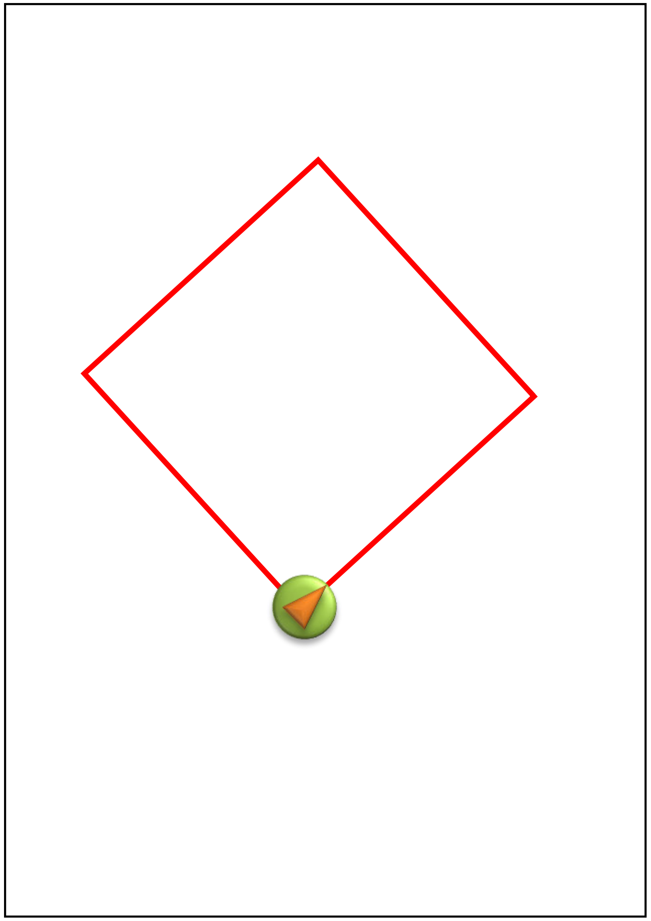
method

control structure

statement

Example — drawing with a Turtle

- Turtle Commands
  - move(dist)
  - turn(angle)
  - penUp()
  - penDown()
  - center()
  - setColor(Color)
- Domain knowledge
  - Starting state
  - Size of world

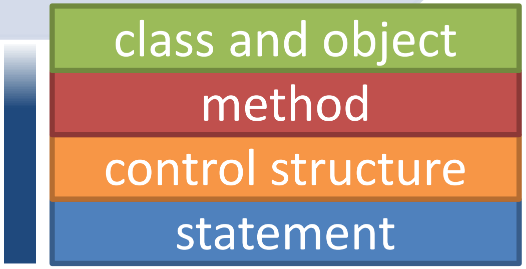


Background information: https://en.wikipedia.org/wiki/Turtle_graphics

‘Low Level’ algorithms

‘Low-level’ algorithms use the constructs of the programming language

- Set aside storage area
- Store value
- Overwrite value
- Arithmetic calculations
- Branch (depending on some property)
- Repeat (depending on some property)



class and object

method

control structure

statement

Activity – Average of two numbers

START

GOAL

a 5

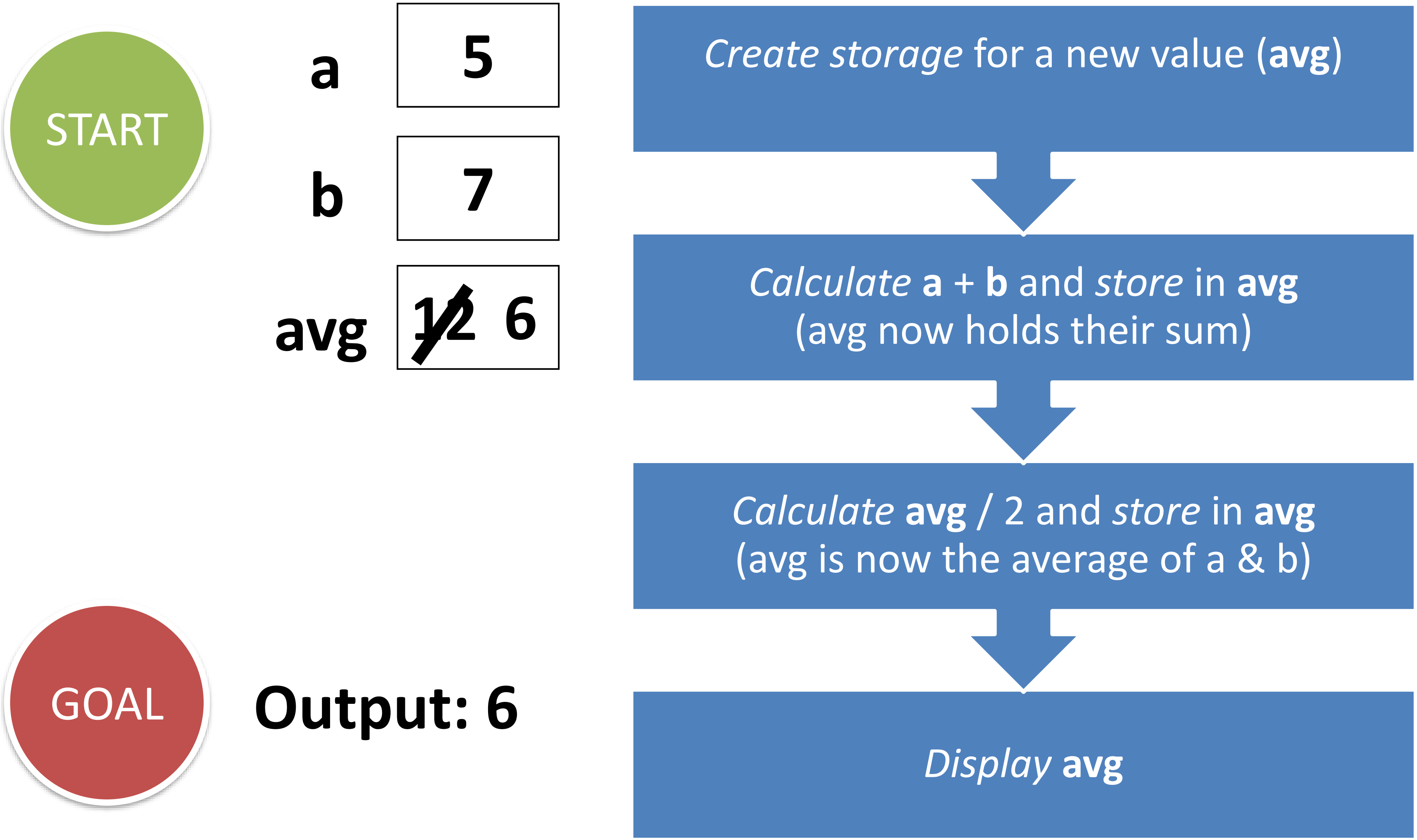
b 7

Display the average value of a and b (which we know in this case is 6)

You can

- create new labelled boxes
- perform arithmetic on box contents and constants & store the result in a box
- display (to the user) the contents of a box

An algorithm to average two values



START

a 5

b 7

avg ~~12~~ 6

GOAL

Output: 6

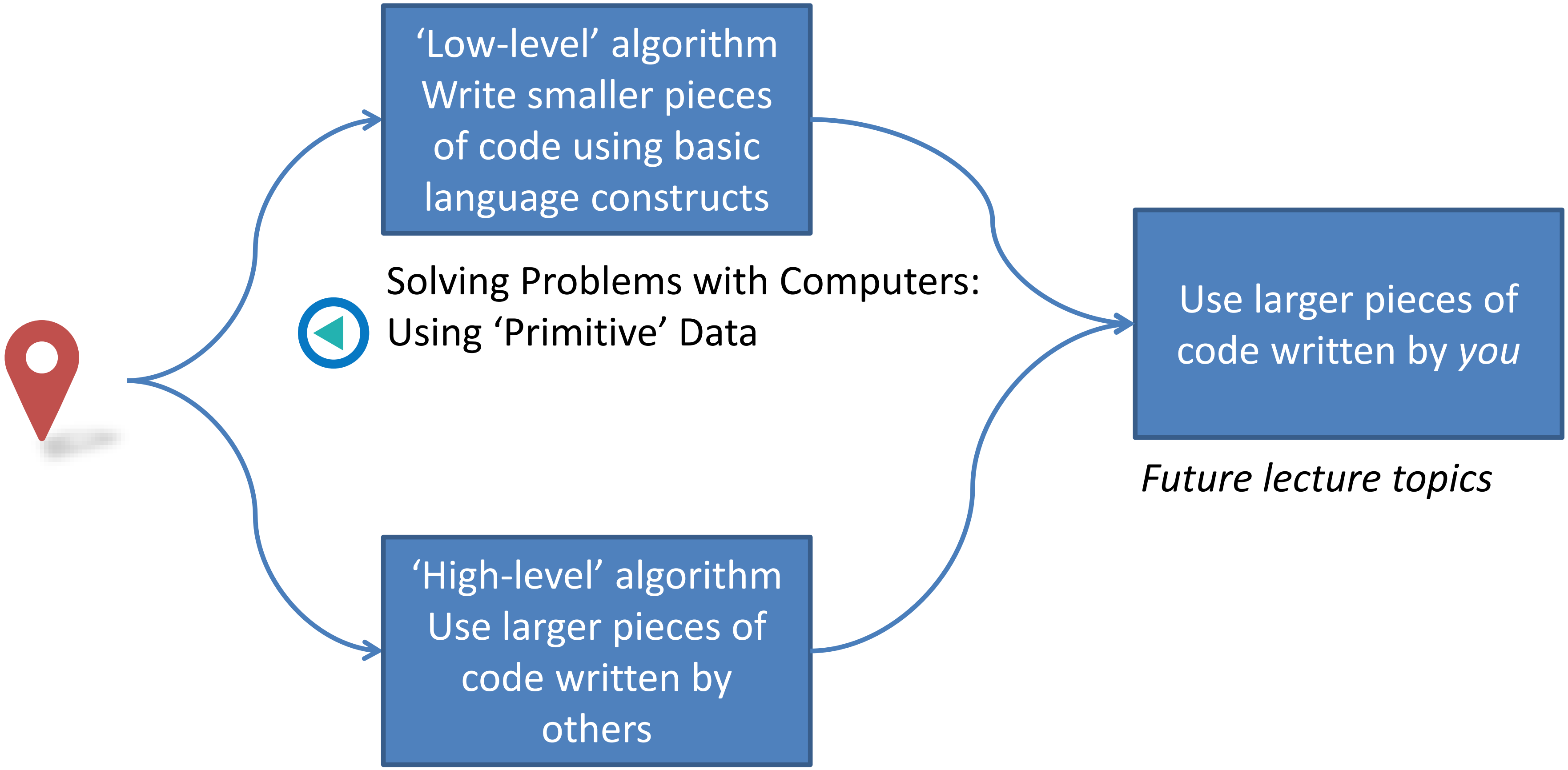
Create storage for a new value (**avg**)

Calculate **a** + **b** and *store* in **avg**
(avg now holds their sum)

Calculate **avg** / 2 and *store* in **avg**
(avg is now the average of a & b)

Display **avg**

Where to next?



‘Low-level’ algorithm
Write smaller pieces of code using basic language constructs

Solving Problems with Computers:
◀ Using ‘Primitive’ Data

Use larger pieces of code written by *you*

Future lecture topics

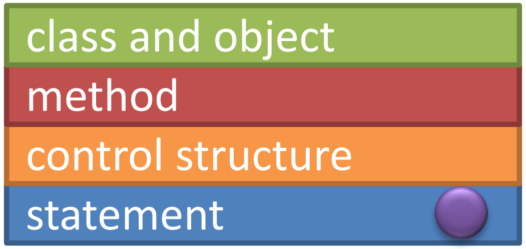
‘High-level’ algorithm
Use larger pieces of code written by others

Solving Problems with Computers:

◀ Importing & Using Objects

◀ Some Commonly Used Objects

Solving Problems with Computers: Using ‘Primitive’ Data

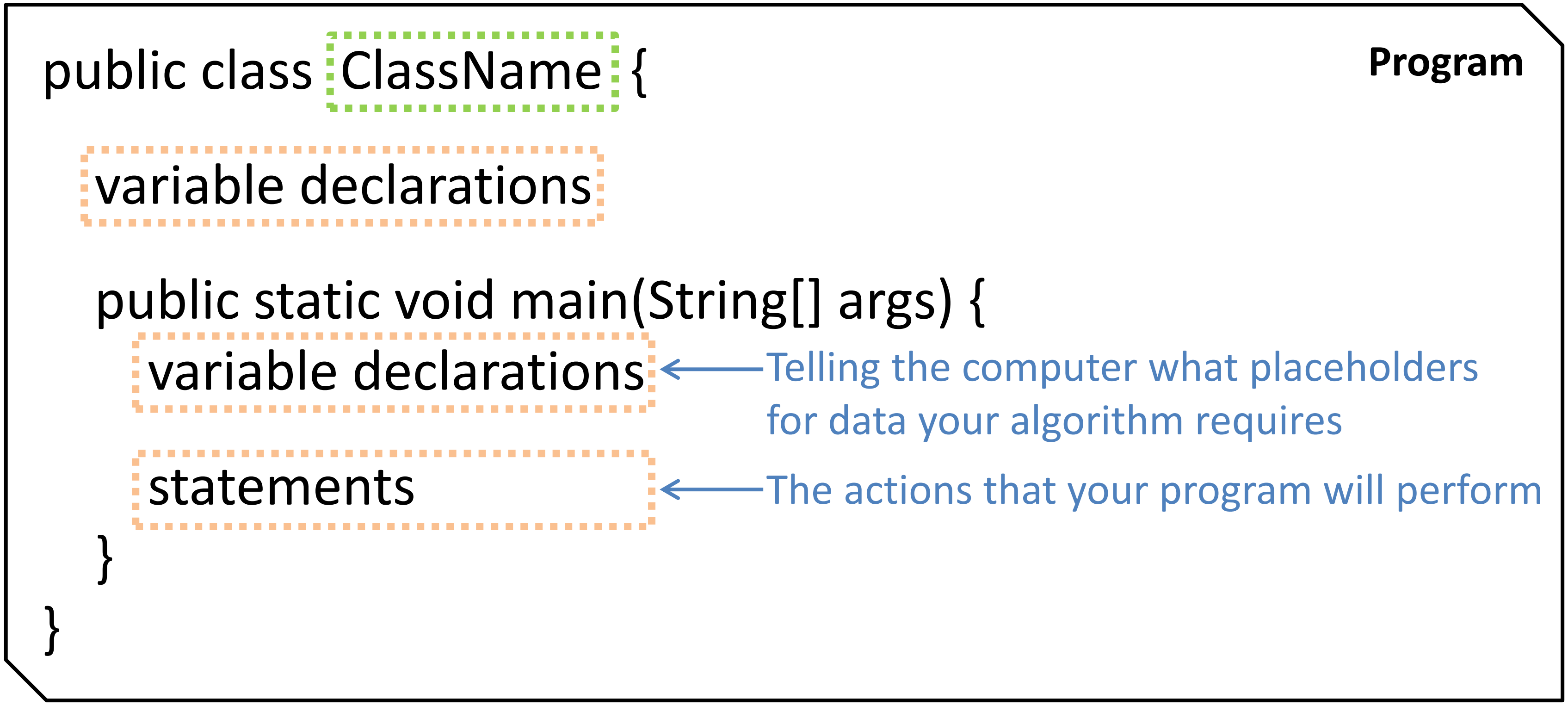

class and object
method
control structure
statement

04 Working with Primitive Data

cc BY NC SA

Dr James Montgomery, james.montgomery@utas.edu.au

Template for a program



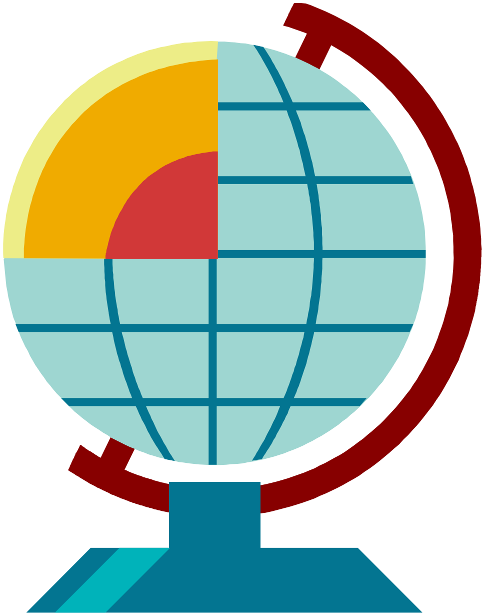
Program

```
public class ClassName {

    variable declarations

    public static void main(String[] args) {
        variable declarations

        statements
    }
}
```

Telling the computer what placeholders for data your algorithm requires

The actions that your program will perform

Programs model the real world



What features or characteristics would you use to model a car?

Data has a type

What values can be represented

“Hello World!”

25

true

3.14159

'a'

What operators can be applied

–

+

/

||

%

&&

Statements and expressions

Statement: a single instruction to the computer

```
System.out.println("Hello");
myTurtle.penDown();
```

Expression: anything that can be *evaluated* to produce a single *value*

```
1 + 1
"Hello"
2
```

Variable declaration

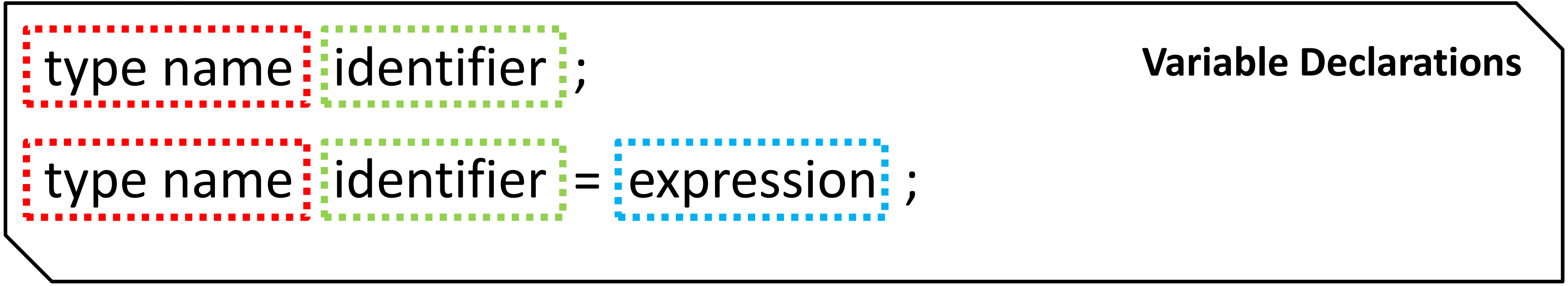
- Planning (pseudocode)

  Variables:

  type identifier, *short description*

  int myAge, *age in years*

- Implementation



type name identifier ;

type name identifier = expression ;

Variable Declarations

```
int myAge; //age in years
double pi = 3.14159;
Turtle drawingTool;
```

Primitive versus object types

Primitive types	Class types (objects)
• data only • *one* piece of data	• data *and* methods (behaviour) • may hold many primitives and other objects

Later in the unit: primitives and objects are stored in different areas of memory

Java primitive number types

Integers

- **byte** 8 bits
  -128 to 127
- **short** 16 bits
  -32768 to 32767
- **int** 32 bits
  -2147483648 to 2147483647
  mostly use this
- **long** 64 bits
  -9223372036854775808 to 9223372036854775807

Floating point (real) numbers

- **float** 32 bits
  7-8 significant digits
- **double** 64 bits
  15-16 significant digits
  mostly use this

Why should we care? *Gangnam Style*
http://www.abc.net.au/news/2014-12-05/gangnam-style-psy-gallops-beyond-youtube-counter/5945606

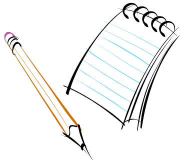
Characters and Booleans

Characters (see Appendix C)

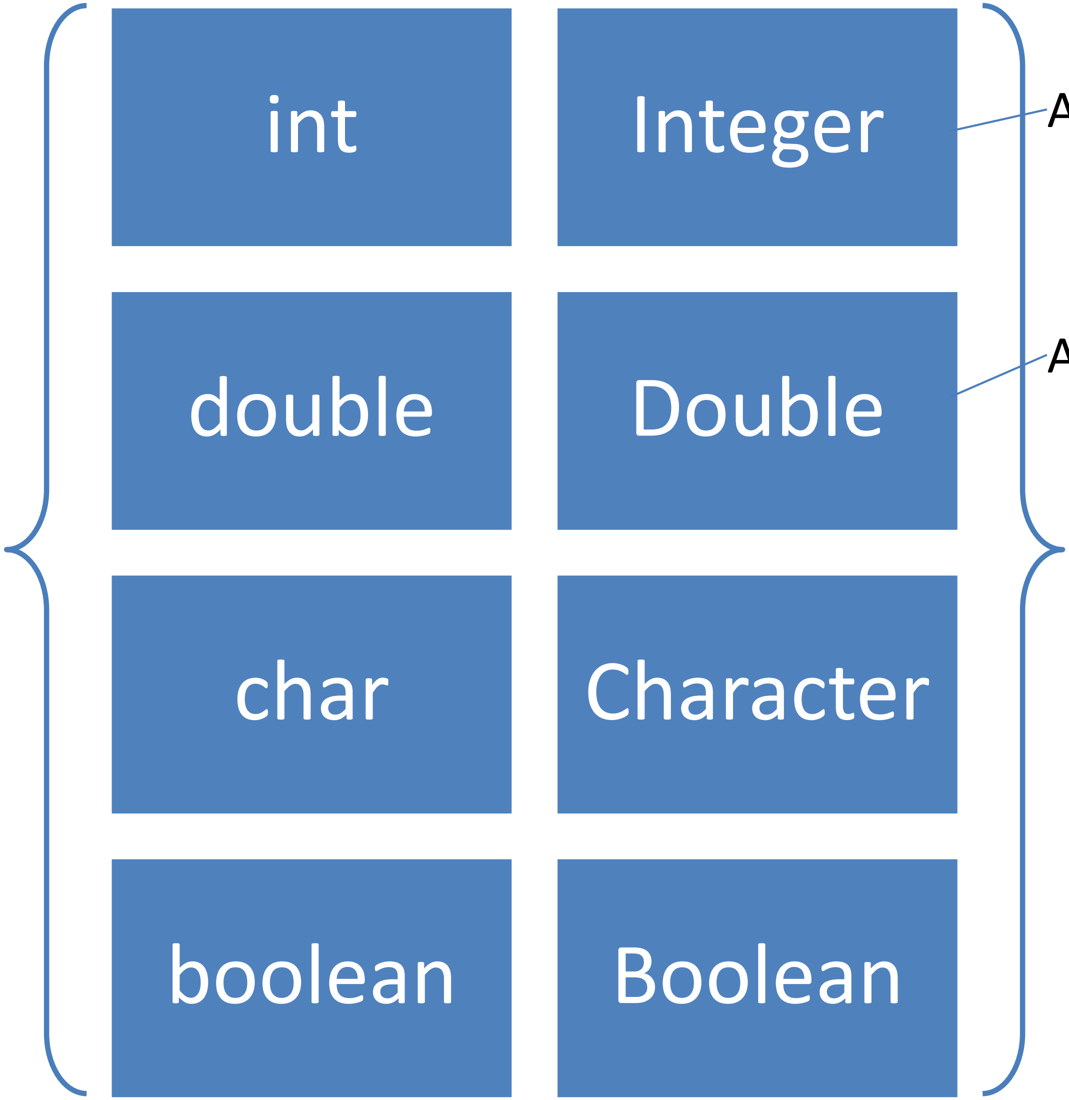
- **char** 16 bits
  any *one* of ~65,536
  Unicode characters
  e.g., 'a' or 'Z' or 'Я' or '子'
- single characters only

Booleans

- **boolean** > 1 bit
  value is **true** or **false**



Primitive wrappers



primitive types	class types
int	Integer
double	Double
char	Character
boolean	Boolean

Also: Byte, Short, Long

Also: Float

class types

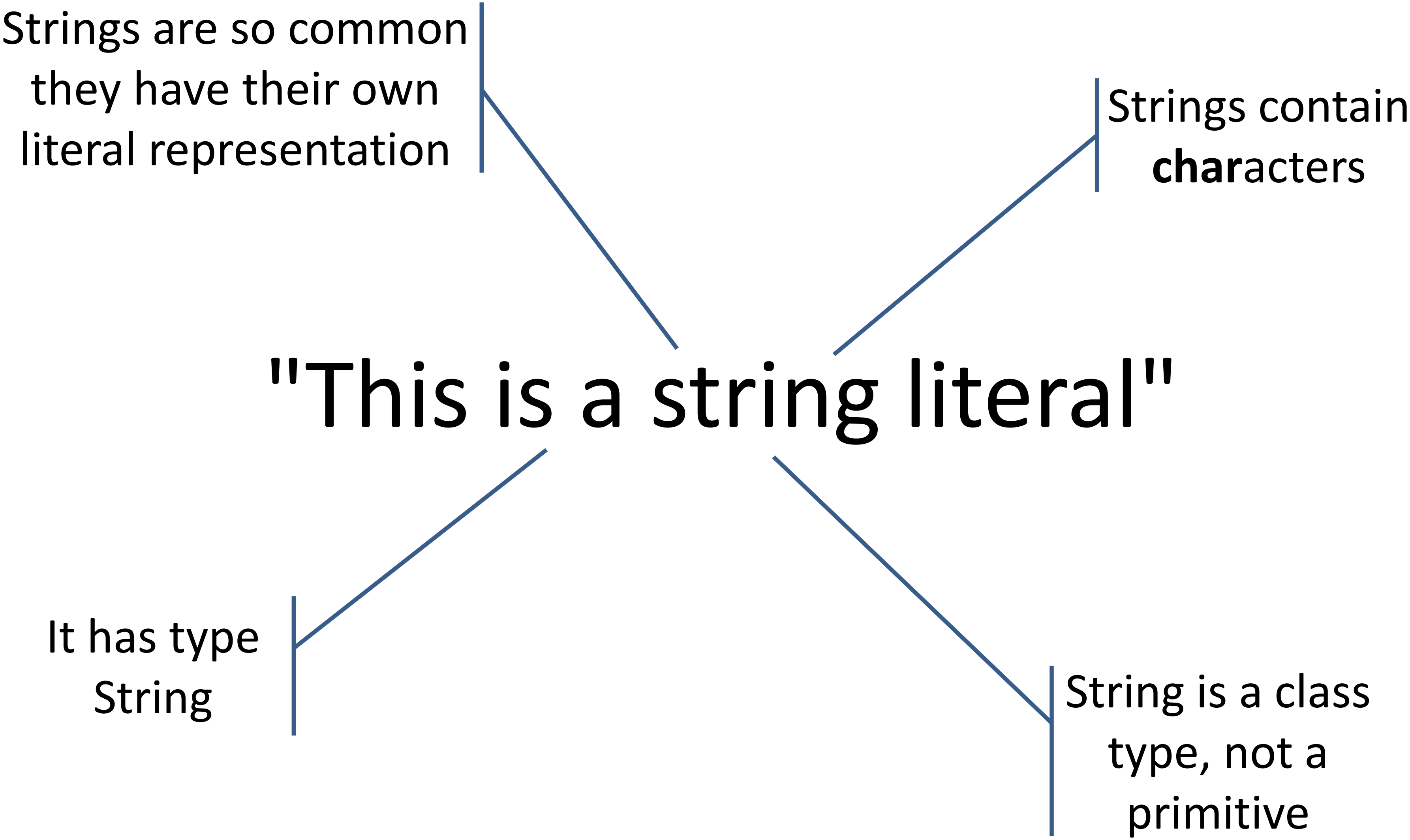
- 'wrap' primitives when an object is needed
- have methods for working with primitives

Value literals

Expressions that literally represent a single value

`int`	• Numerals with no decimal point • 1 2 1024 etc.
`double`	• numerals with a decimal point • 1. 1.0 2.5 3.14159 etc.
`char`	• character in single quotes • 'a' 'z' '1'
`boolean`	• true false

A word about `Strings`



Strings are so common they have their own literal representation

Strings contain **characters**

"This is a string literal"

It has type String

String is a class type, not a primitive

Assignment

Changes the value of a variable by assigning it the value of an expression

- Planning (pseudocode)

  identifier = **expression value** Read = as *becomes*

  myAge = 18

- Implementation

identifier = expression ;

Variable Assignment

```
myAge = 18;
pi = 3.14159;
drawingTool = new Turtle();
```

Type of expression must be compatible with type of variable

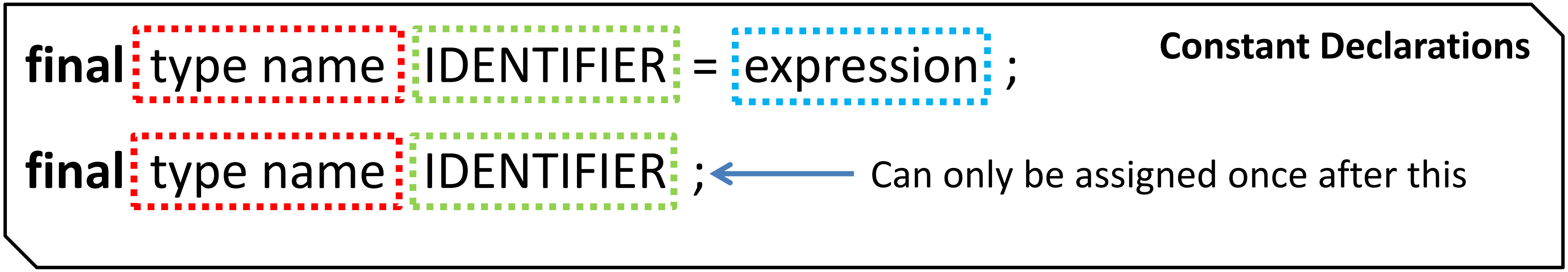
Constants

365 _____ in a _____

12 _____ in a ______

24 _____ in a _______

A constant is a named value (i.e., a variable) whose value, once assigned, cannot be changed during program execution



Constant Declarations

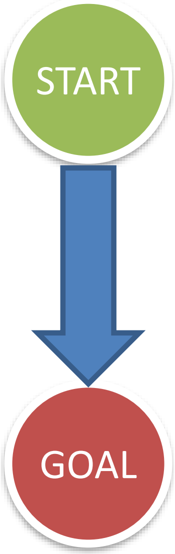
final type name IDENTIFIER = expression ;

final type name IDENTIFIER ; ← Can only be assigned once after this

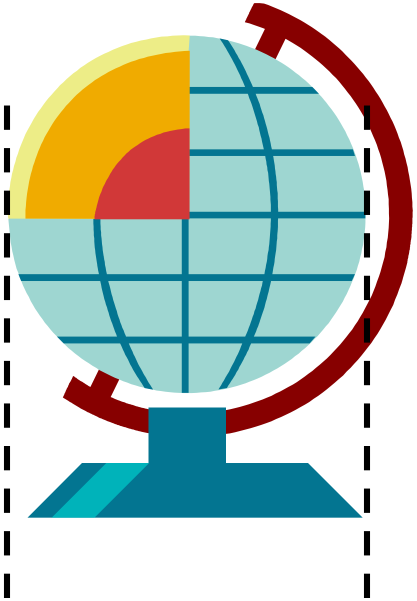
Summary

We can solve problems with algorithms…

…that model some aspects of the problem with different types of data


START

GOAL



```
int population;

double birthRate;

boolean isWarming;

char type = 'E';

final double DIAMETER = 12742;
```

Solving Problems with Computers: Importing and using objects

- class and object
- method
- control structure
- statement ●

05 Using Objects

cc BY NC SA

Dr James Montgomery, james.montgomery@utas.edu.au

The story so far

Programs instruct computers to perform actions

- *manipulate data*
- *display graphics*
- …

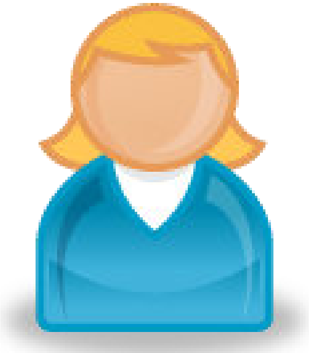
well-suited to implementation as

Algorithms are sequences of steps for solving problems

Programs *model* aspects of the real world

- *person’s height*
- *document’s text*
- …

Java primitive types can model **whole** and **real numbers**, individual **characters**, and **Boolean** values



```
int age;
char gender;
int height;
double weight;
boolean wearsGlasses;
```

What if we wanted to model > 1?

```
public class PersonDatabase {
    public static void main(String[] args) {
        int age;
        char gender;
        int height;
        double weight;
        boolean wearsGlasses;

        //Lots of code to store values
        // and do something with them
    }
}
```

Only good enough for one person

Primitive versus object types

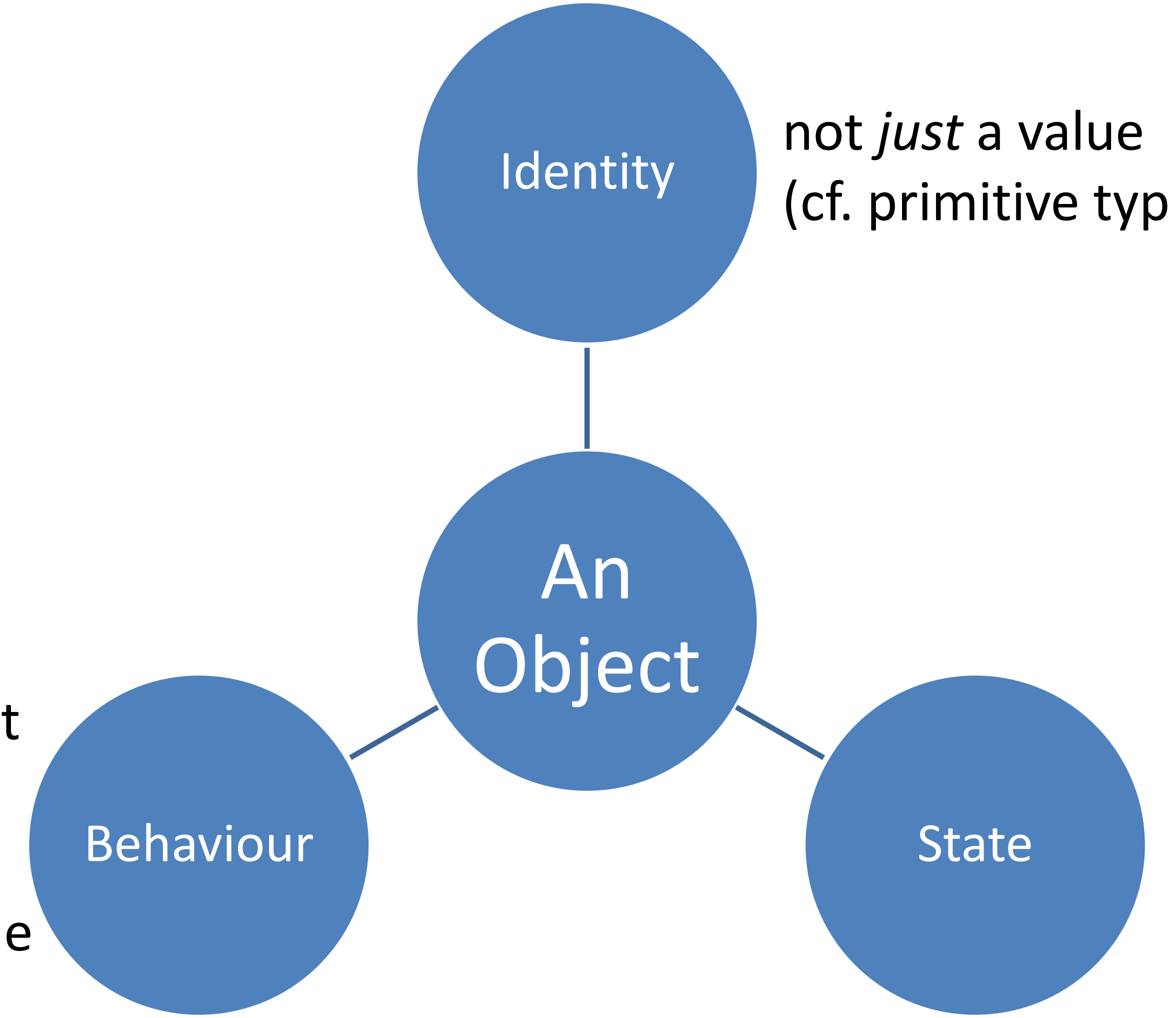
Primitive types

- data only
- *one* piece of data

Class types (objects)

- data *and* methods (behaviour)
- may hold many primitives and other objects

Characteristics of an Object



An Object

Identity

not *just* a value
(cf. primitive types)

Behaviour

methods that can affect its own state or perform some action

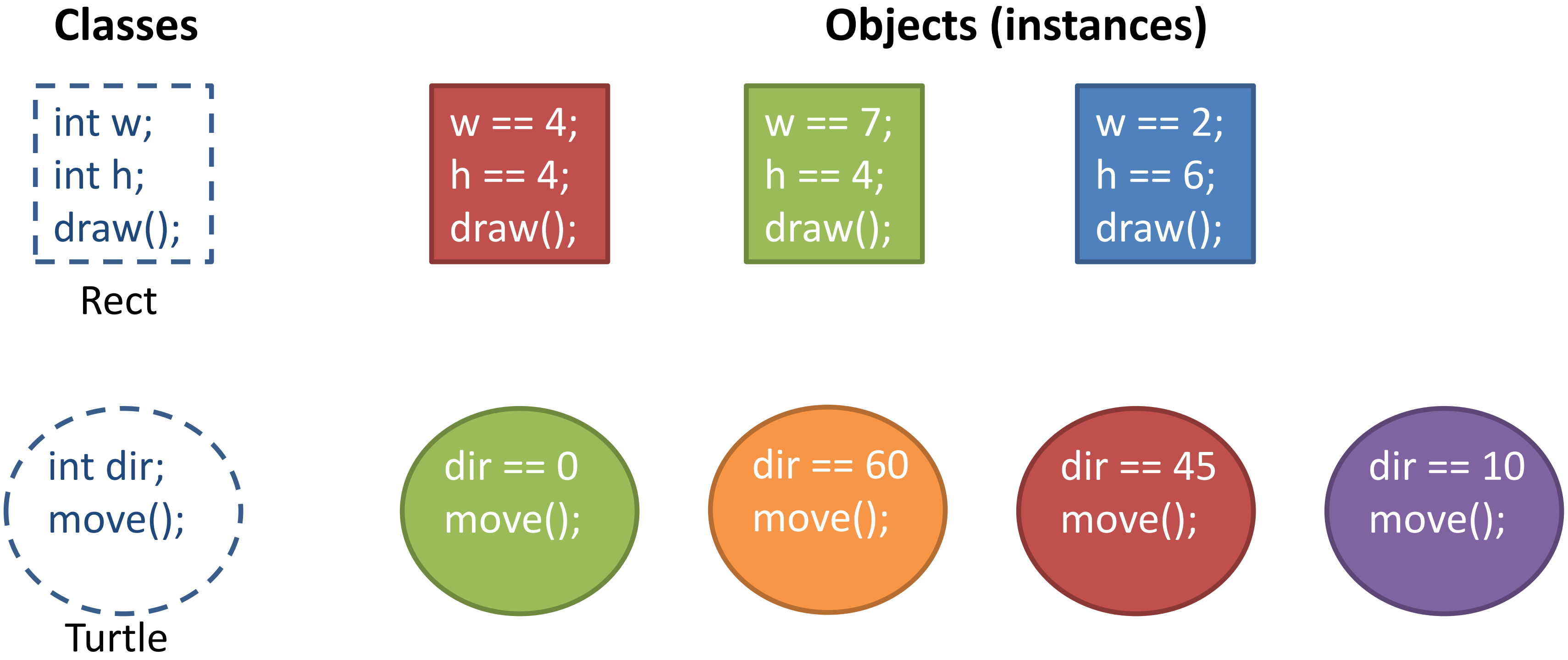
State

current **values of its variables,** known as attributes or members

Classes (of objects)

Abstractly: Objects with the same set of properties and with the same behaviours/abilities form a *class*

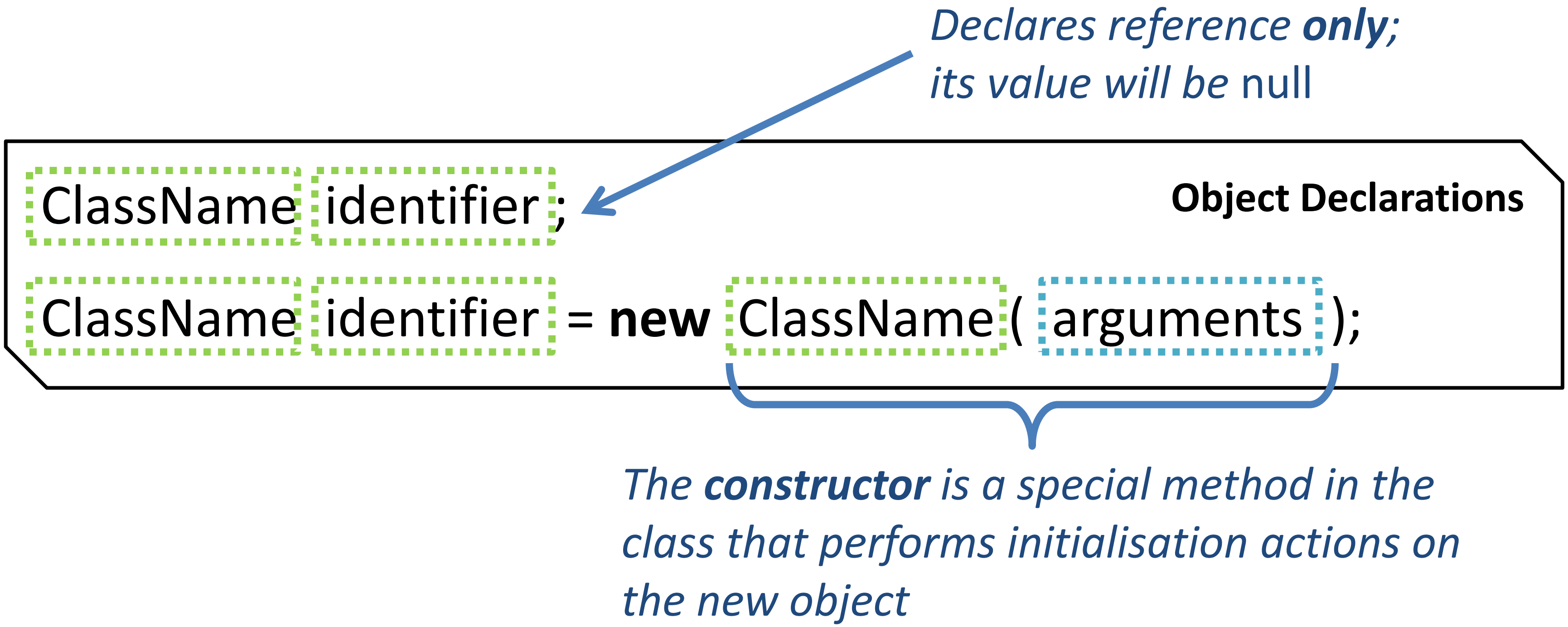
In practice: Source code for **one** class defines **all** the objects (called instances) of that class



Creating Objects

A variable either holds a primitive type, or it holds a **reference** to, i.e. the address of, an object

Actual object is created with **new** keyword



Declares reference ***only****;*
its value will be null

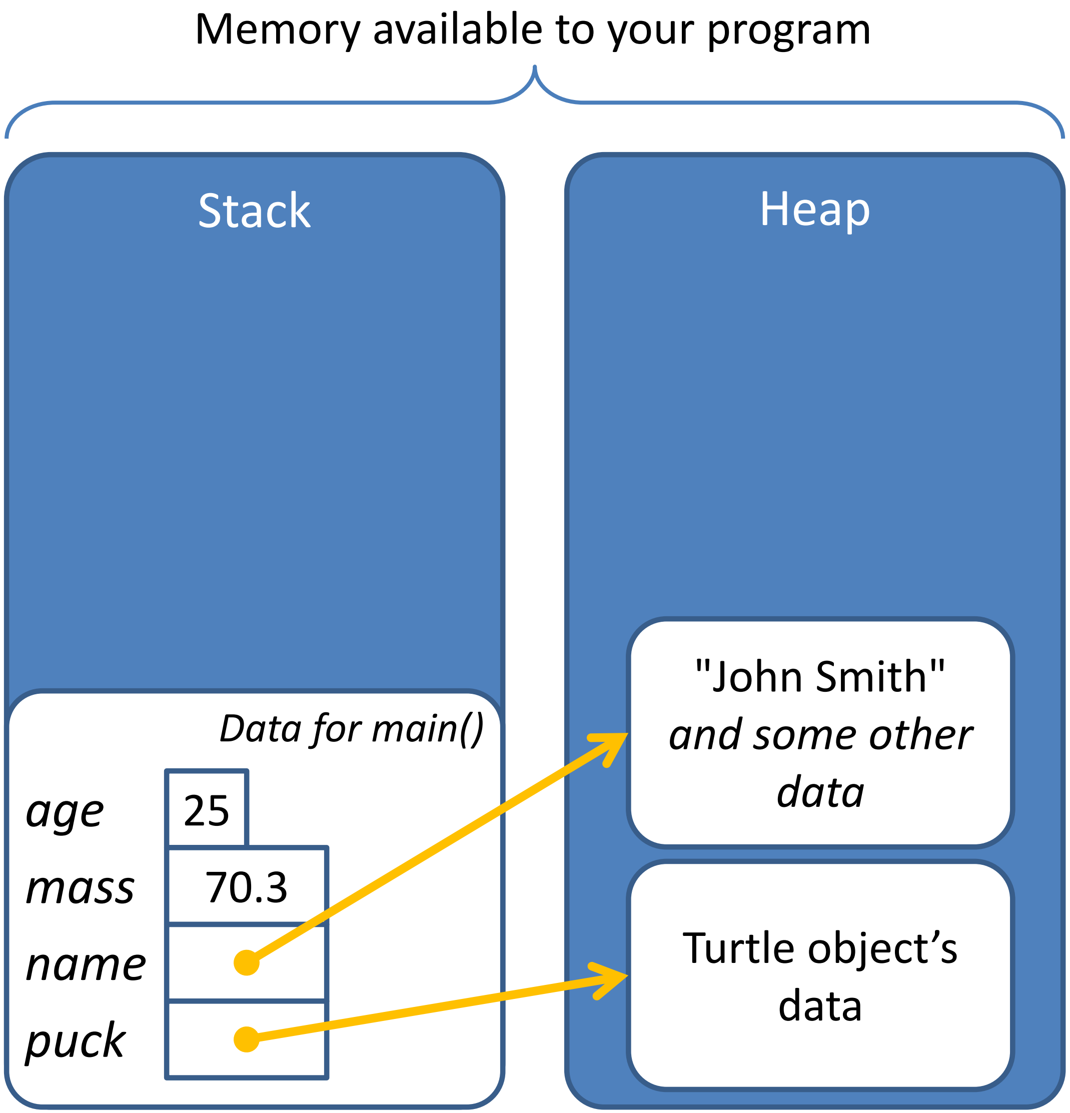
Object Declarations

```
ClassName identifier;
ClassName identifier = new ClassName( arguments );
```

The ***constructor*** *is a special method in the class that performs initialisation actions on the new object*

Behind the Scenes: Heap v Stack

```
public class Memory {
  public static void main(String[] args) {
    int age = 25;
    double mass = 70.3;
    String name = "John Smith";
    Turtle puck = new Turtle();
  }
}
```



Memory 'boxes' scaled according to number of bits.
This assumes a 64-bit machine, so memory addresses are 64 bits long.

Examples

```
String title;
Turtle fred;
Scanner sc;

title = new String(); //creates a new, empty string
fred = new Turtle();
//Scanner's constructor takes an argument
sc = new Scanner( System.in );

//Or

Turtle arthur = new Turtle();
//arthur is a different Turtle object to fred
```

A sample complete program

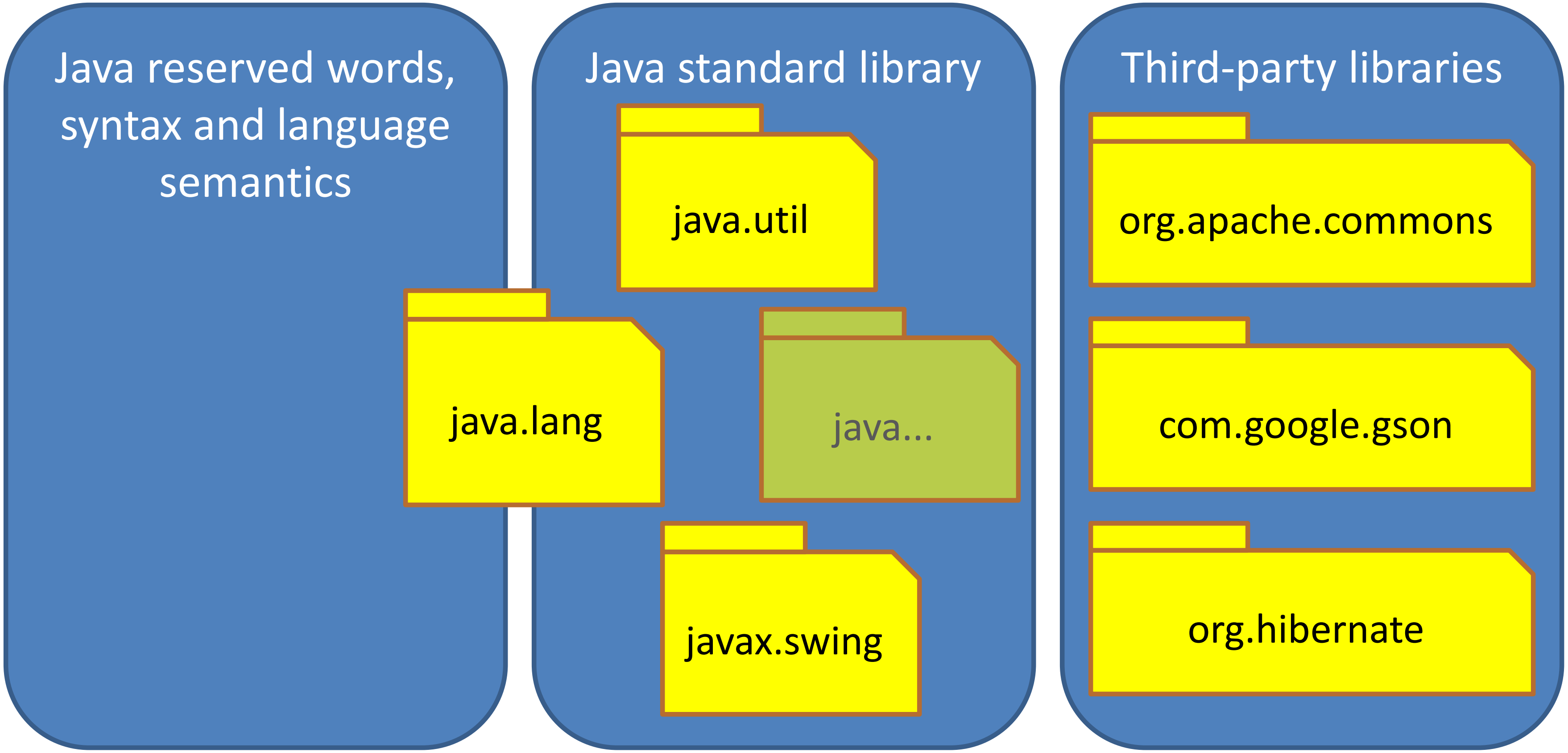
```
import java.util.Scanner;

public class CreateAnObject {

    public static void main(String[] args) {
        String word;
        Scanner sc = new Scanner(System.in);


        word = sc.next();
        System.out.println("The word was " + word);
    }

}
```

Declaration

Class names

Declaration and instantiation (creation)

Java *language* versus *libraries*

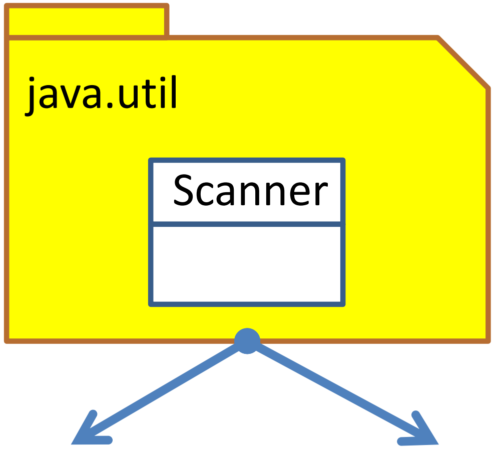


Related classes are grouped into packages (and subpackages)

Other programming languages have similar mechanisms

importing classes

To use a class defined in some package…



Option 1: Use fully-qualified name in declaration and instantiation

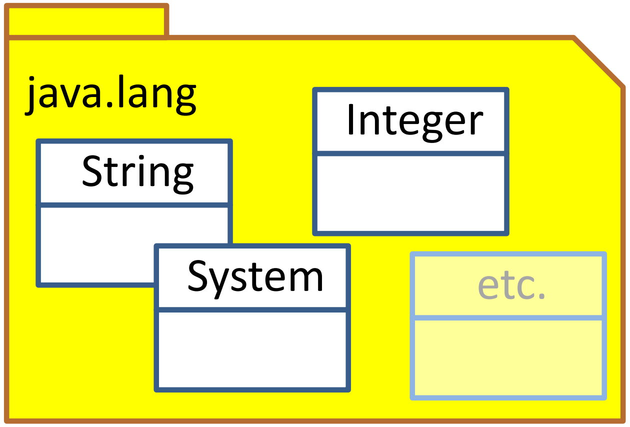
```
...
java.util.Scanner sc;
sc = new java.util.Scanner(System.in);
...
```

Option 2: Import it

```
import java.util.Scanner;

public class WithImport {
   public static void main(String[] args) {
      Scanner sc;
      sc = new Scanner(System.in);
   }
}
```

java.lang package and import wildcards

Contents of `java.lang` package always available



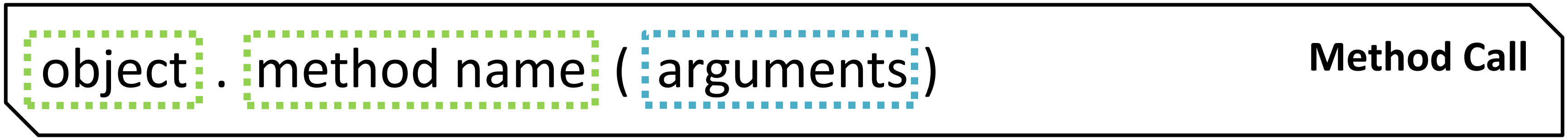
Can import everything from a package with *

e.g., `import java.util.*;`

```
import kit101.turtle.*;
```

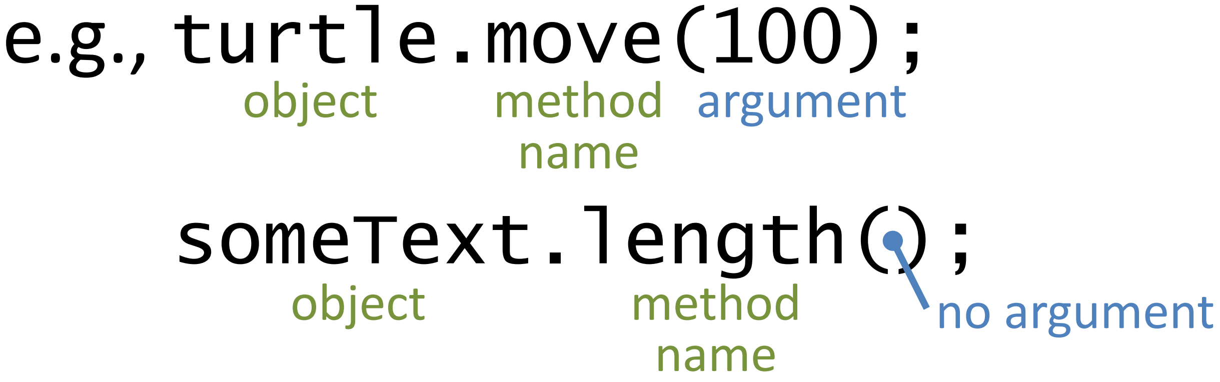
(but better practice to import only what you need)

Method call



object . method name (arguments)

Method Call



e.g., `turtle.move(100);`

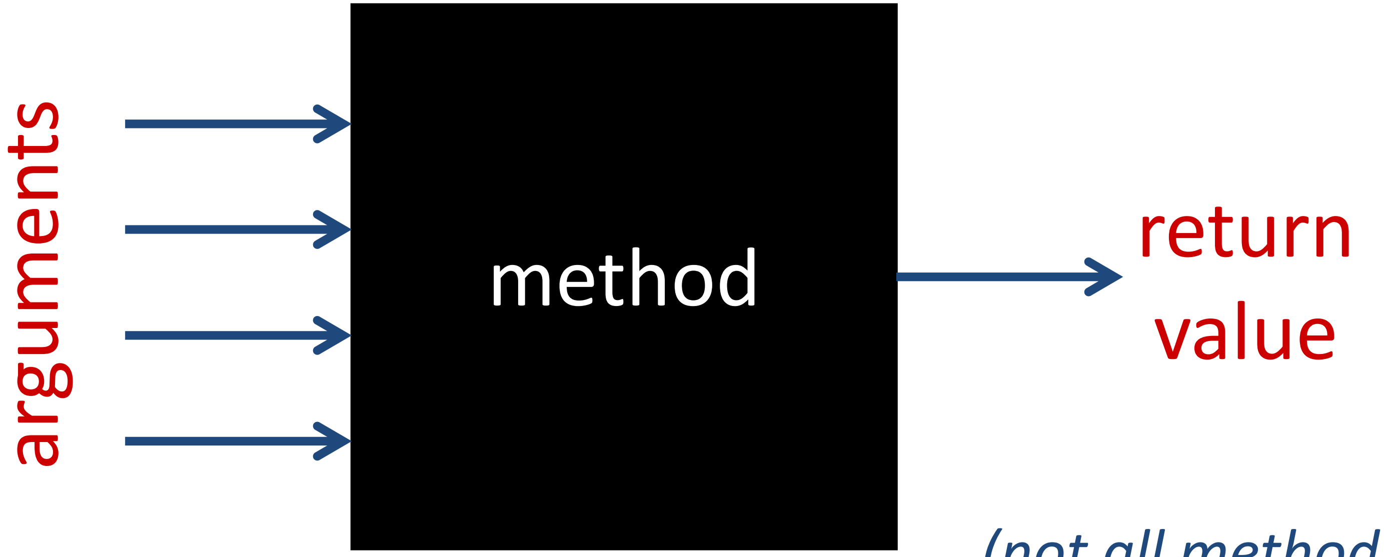
`someText.length();`

In general

- Methods can take zero or more arguments
- If a method returns a value then must be assigned to a variable if needed later, as in

```
int origLength = someText.length();
```

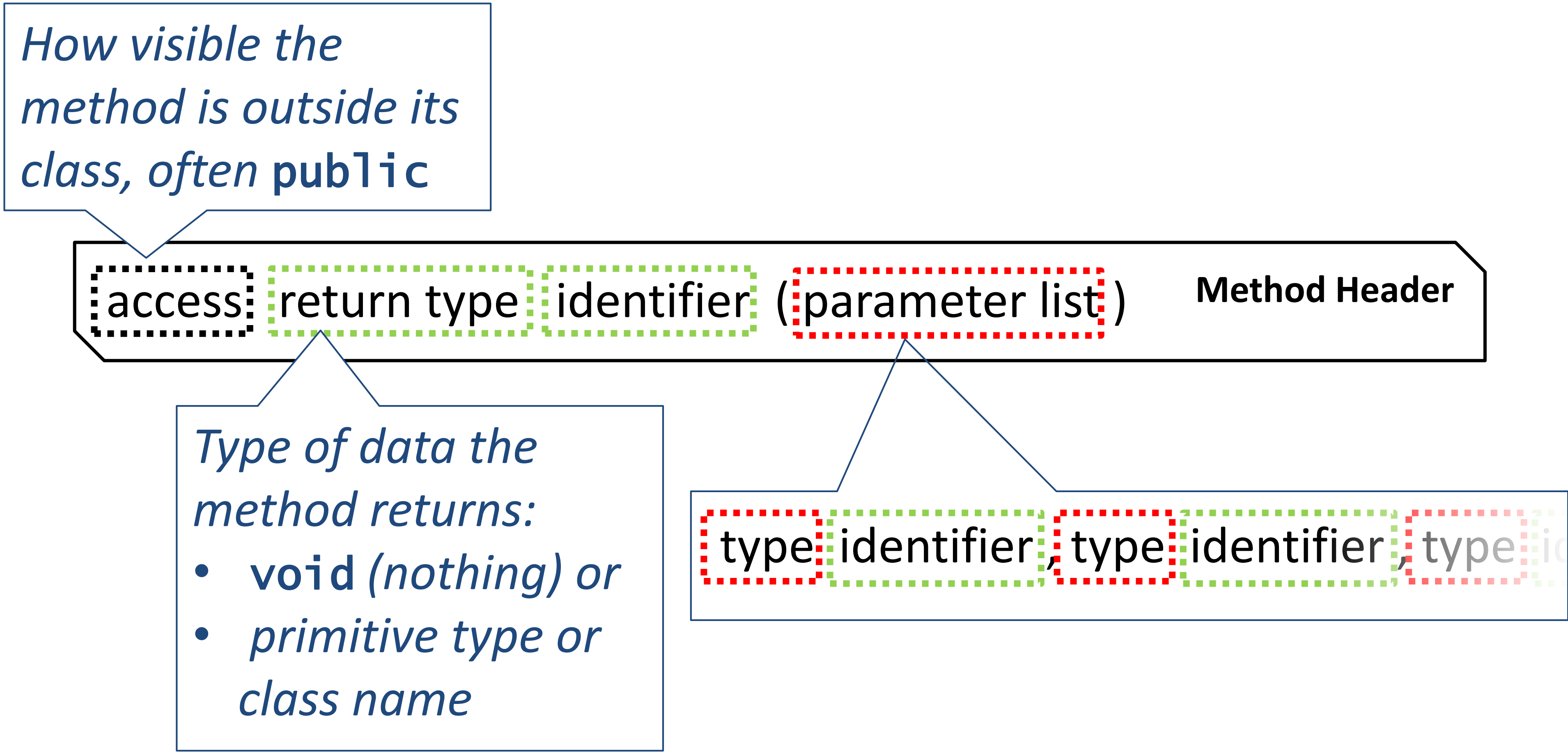
Treat methods as black boxes



(not all methods return a value)

The method header

First line of a method declaration



How visible the method is outside its class, often **`public`**

Type of data the method returns:

- **`void`** *(nothing) or*
- *primitive type or class name*

Solving Problems with Computers: Some commonly used objects

plus one we provide for practice

- class and object
- method
- control structure
- statement ●

05 Using Objects

cc BY NC SA

Dr James Montgomery, james.montgomery@utas.edu.au

Some (of many) useful Java classes

From the Java Class Library

- String of characters (i.e., text)
- Scanner for reading user input
- Random number generator
- Math utilities
- System utilities (& text output)

From this unit

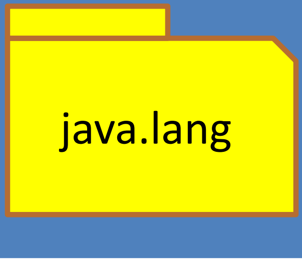
- Turtle graphics

Click the class name to go to that slide

String class

Import

- Not required


java.lang

Instantiation

```
String s = new String("not necessary");
String s2 = "can use a literal";
```

Special features

- immutable (cannot be changed)
- concatenation (joining) operator +
- can access character at each position, starting from 0

Useful methods

```
char charAt(int i)
int indexOf(char c)
int compareTo(String s)
boolean equals(String str)
String substring(int startAt,
                 int endBefore)
```

see Strings.java*

Scanner class

Import

import java.util.Scanner;

Instantiation

Scanner sc;

sc = new Scanner(System.in);

System.in is a stream of characters from ‘standard input’; often the keyboard

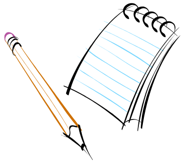
Warnings

- *nextType* methods read next value of that *type* up to whitespace
- next() reads next word (up to next whitespace)
- nextLine() reads *entire* line, including any whitespace

Useful methods

```
int nextInt()
double nextDouble()
boolean nextBoolean()
String next()
String nextLine()
```

Combining object abilities



Problem: Read and store a single character typed by the user

(given that the **Scanner** class does not have a **nextChar()** method)

see CharEntry.java*

Random class

Import

```
import java.util.Random;
```

Instantiation

```
Random rand = new Random();
Random rand2 = new Random(123);
```

123 is a random seed

Pseudorandom

- Computers do not generate truly random numbers but sequences of numbers that are sufficiently close to random
- Setting the random seed allows the same sequence to be produced

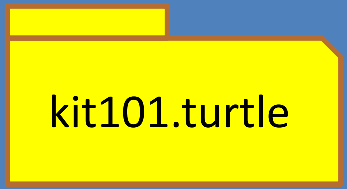
Useful methods

```
int nextInt(int limit)
double nextDouble()
void setSeed(long seed)
```

Turtle class

Import

```
import kit101.turtle.Turtle;
```



kit101.turtle

Requires that kit101 and turtle folder are in same folder as your program

Instantiation

```
Turtle t = new Turtle();
```

Initial state

- position (centre of world)
- direction (facing east)
- pen down? (true)
- pen colour (black)

Useful methods

```
void move(double dist)
void moveTo(int x, int y)
void turn(double deg)
void penUp()
void penDown()
void setColor(java.awt.Color c)
```

Random Turtles

Task: Modify **`TurtleStart.java`** so the **`Turtle`** draws squares of a random size 10–100

Plan:

1. import **java.util.Random**
2. declare and instantiate a **Random** object
3. get a random number 0–9
4. add 1, then multiply by 10 (gives a number 10–100)
5. use this number as the side of the square

see Turtle.java*

Class (static) members

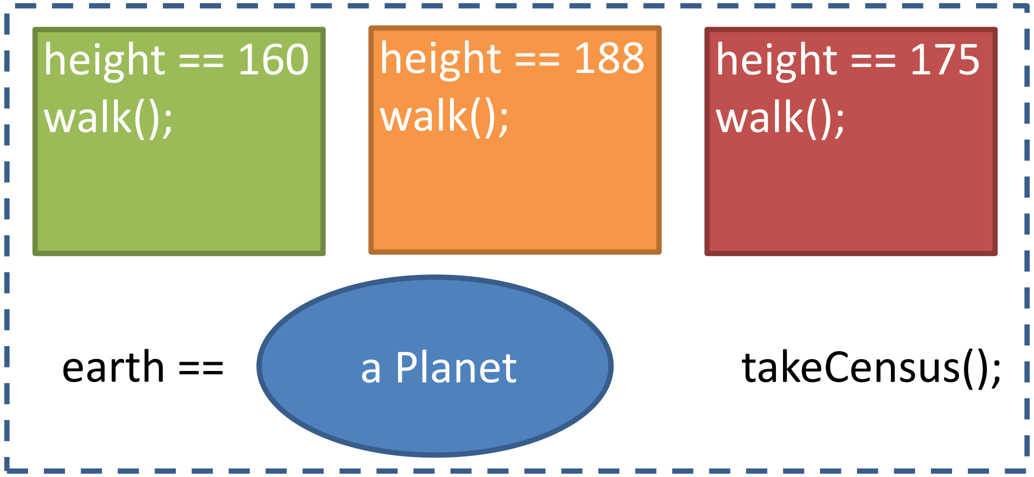
Objects of a class may share a single copy of some data and some methods

Classes

```
int height;
static Planet earth;
void walk();
static int takeCensus();
```

Person

Objects (instances)



height == 160
walk();

height == 188
walk();

height == 175
walk();

earth == a Planet

takeCensus();

```
static final double PI = …
static double cos();
static double sin();
static double max();
```

java.lang.Math

No objects of class Math

PI == 3.14159…..

cos(); sin(); max();

Math class

Import

Not required

java.lang

Instantiation

Not possible

A utility class

- Math functions
- Mathematical constants:
  - π Math.PI
  - e Math.E

Useful methods (lots more)

```
int abs(int num)
double abs(double num)
int max(int n1, int n2)
double sin(double angle)
```

Class Data (Static Data)

Class . method name (arguments) **Static Method Calls**

Class . variable name **Static Data Access**

Examples

- **`Math`** class contains the constant **`PI`** (a **`double`**): `Math.PI`
  - stores the value of the mathematical constant π (a double-precision ‘real’ number)
- **`Color`** class
  - **`import java.awt.Color;`**
  - **`Color.RED`**, **`Color.GREEN`**, etc.
  - these constants are objects of the **Color** class; can be used by some methods, e.g., **`setColor()`** in **`Turtle`** objects)

see MathUsage.java*

System class

Import

Not required

java.lang

Instantiation

Not possible

A utility class; includes

- `static final` streams for standard input, standard output, and standard error: System.in, System.out & System.err
- System.out is **PrintStream** object

Some System.out methods

```
void print(String s)
void println(String s)
void println(int n)
void println(double d)
void println(boolean b)
void println(char b)
```