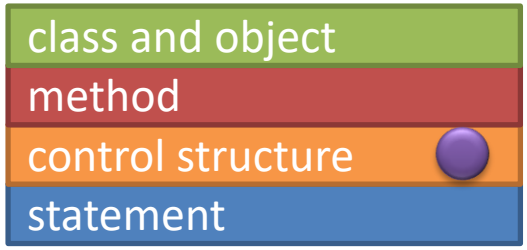
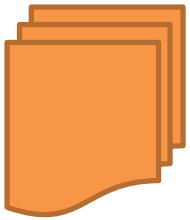
Making Decisions

Part 1: Boolean expressions and comparing values





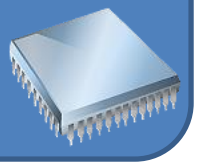
08 Making Decisions

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

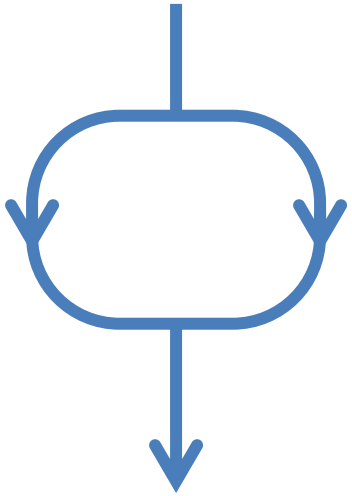
- *with combinations of primitive and class data types*

Programs can be a sequence of statements

1. *Do this*
2. *Do that*
3. *Do this (again)*

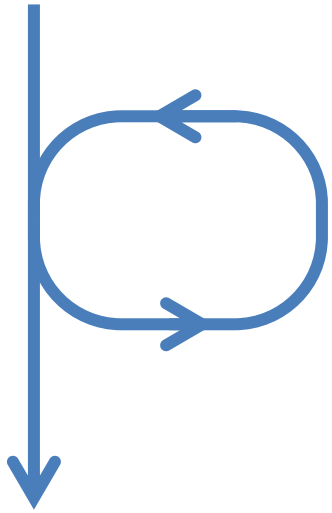
What's missing?

Selection



These slides

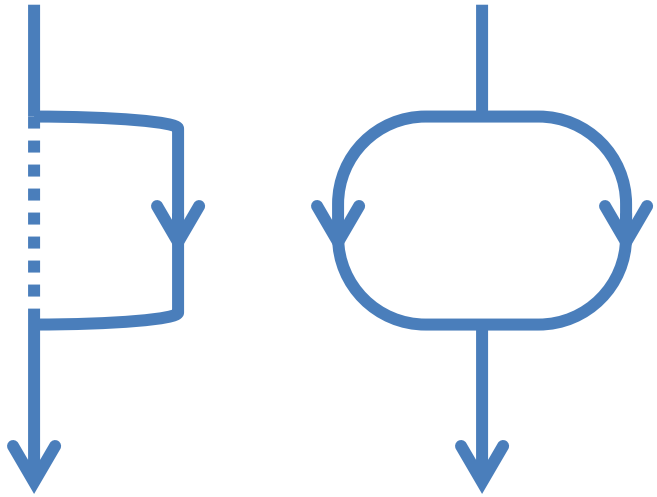
Iteration



Other slides

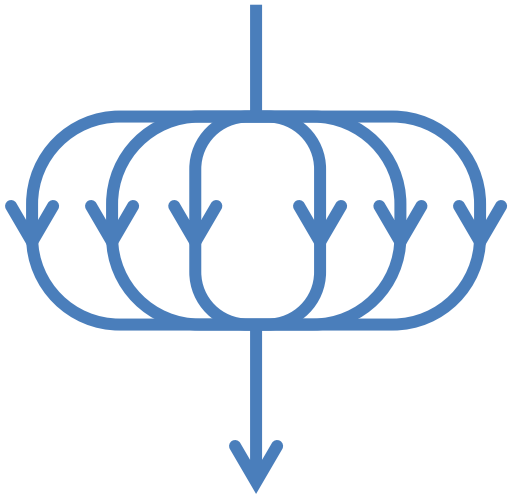
Control structures

`if` `if-else`



Select a path based on value of a **Boolean expression**

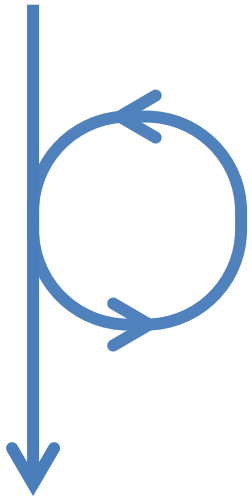
`switch`



Paths ‘labelled’ by a **value** (integer-valued primitives, Enums and Strings)

Loops

`while`
`do-while`
`for`



Repeat actions while **Boolean expression** is true

Comparison Operators

Many Boolean expressions result from comparing one value with another

Equality operators

==	equal to
!=	not equal to

Relational operators

<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to

Comparing… primitives

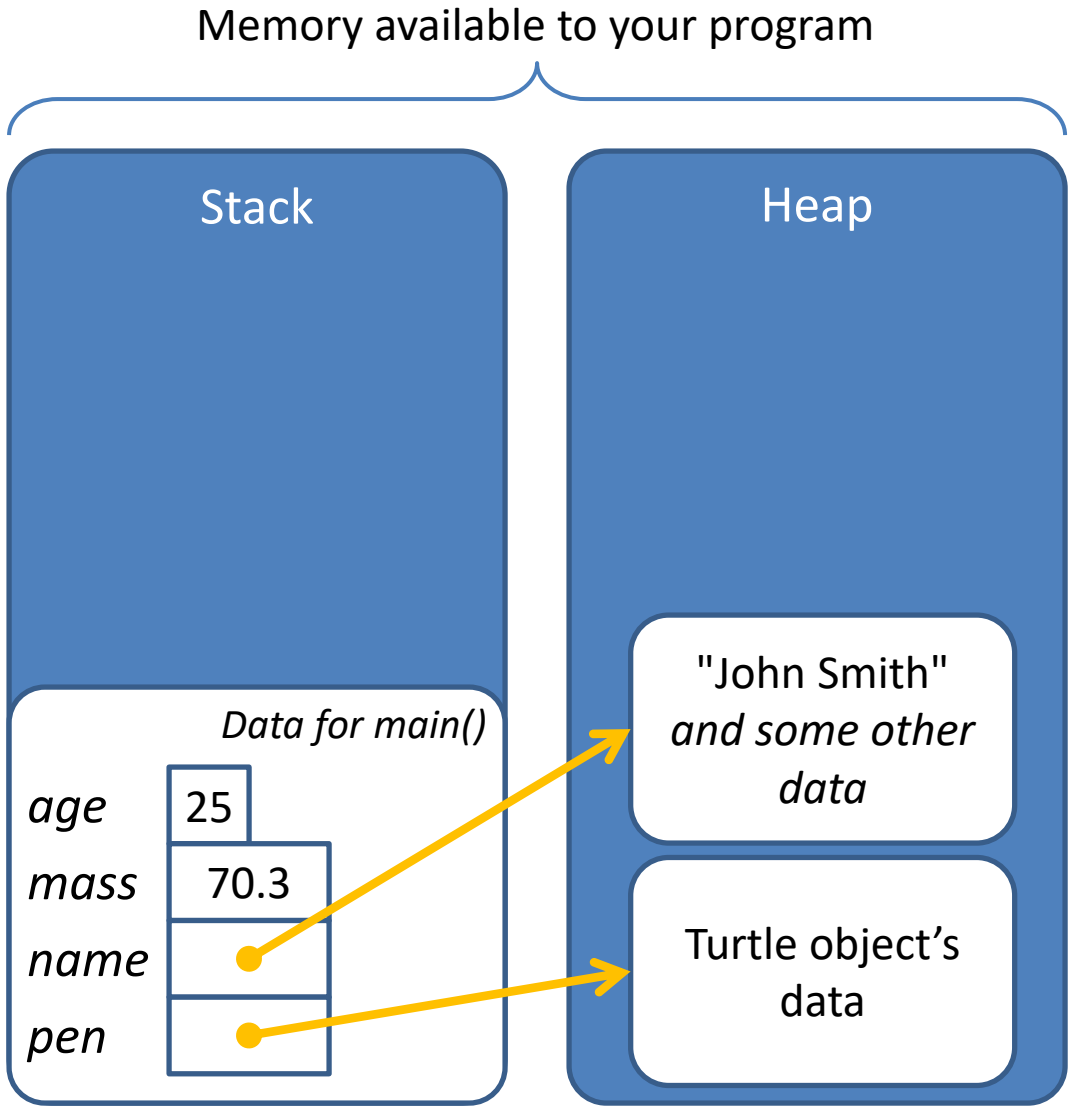
- `int` (`byte`, `short`, `long`): easy
- `char`: based on position in Unicode table, so
  - ‘a’ < ‘z’, but
  - ‘A’ < ‘a’ (and ‘Z’ < ‘a’)
- `double` (and `float`) are not stored precisely
  - Rather than ==, check if *difference* is ‘sufficiently small’
  - Math.abs(d1 – d2) < 0.00001

Degree of tolerance
is up to you

see DoubleTrouble.java

Comparing… objects

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



Comparison operators only work on 'primitive' data: ***primitive types*** *and object* ***references***

Comparing… objects

- All objects have this method:
  `public boolean equals(Object o)`
  - In some, no better than ==
  - In others, returns `true` if internal *state* is same
  - Usage: `someObject.equals( anotherObject );`

- Strings (and other Comparables) also have
  `public int compareTo(String s)`

```
"alpha".compareTo("zeta") == -25
"alpha".compareTo("alpha") == 0
"zeta".compareTo("alpha") == +25
```

Any result < 0 means the first object belongs before the second, any result > 0 means it belongs after it

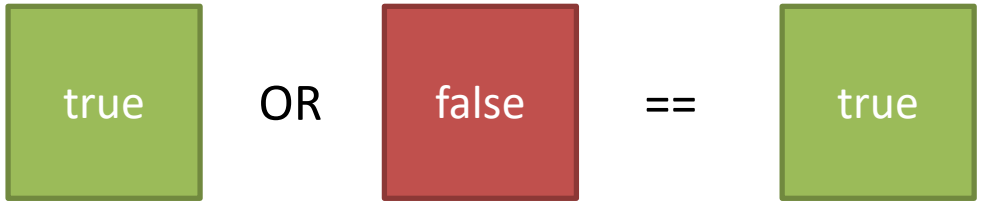
see StringComparison.java

Boolean Operators

Java | *Concepts*

&& AND: both (boolean) operands must be true

| | OR: either (boolean) operand must be true



! NOT: negate/flip/invert a boolean value

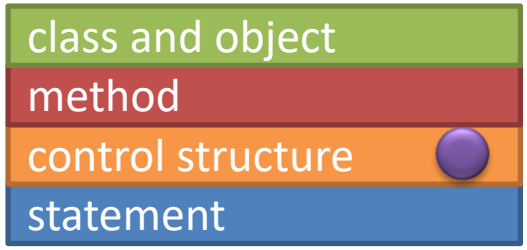
Operator Precedence (for reference)

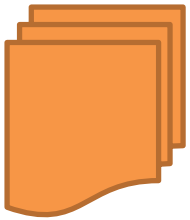
! + –	negate, unary plus or minus
* / %	multiply, divide, modulo (remainder)
+ –	add, subtract
== !=	is equal to, is not equal to
< <= > >=	less than, less than or equal to, greater than, greater than or equal to
&&	logical *and*
||	logical *or*

Parentheses () can always be used to override the default precedence or make clear what is intended

Making Decisions

Part 2: Two-way branching with **if** and **if-else**

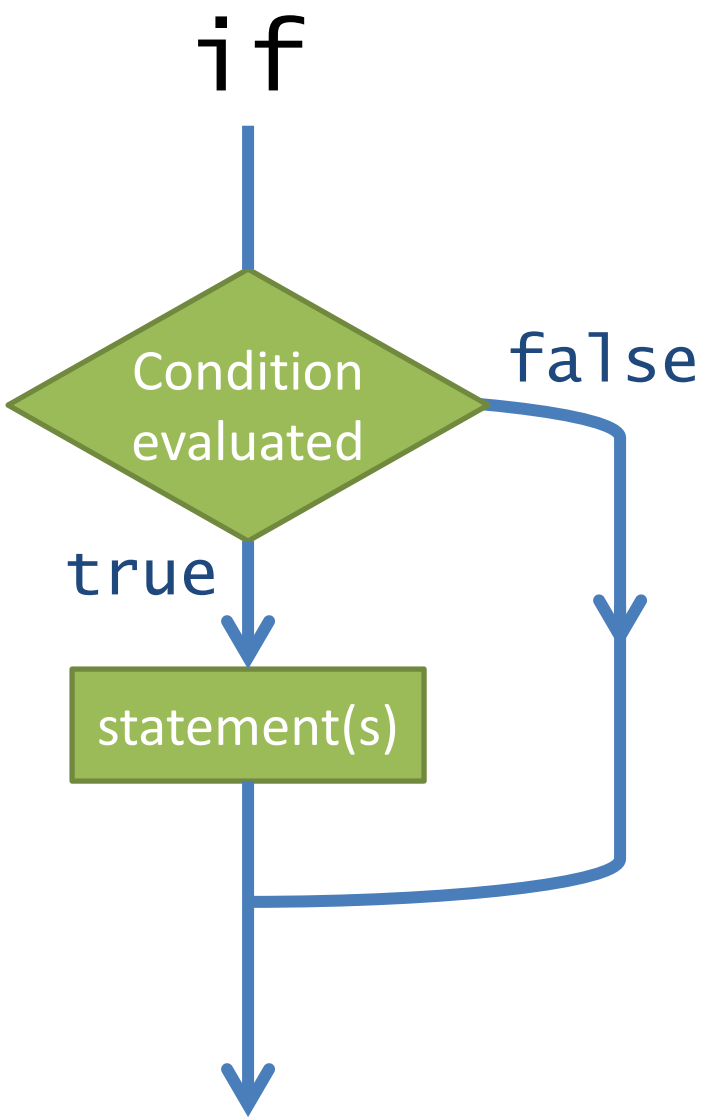


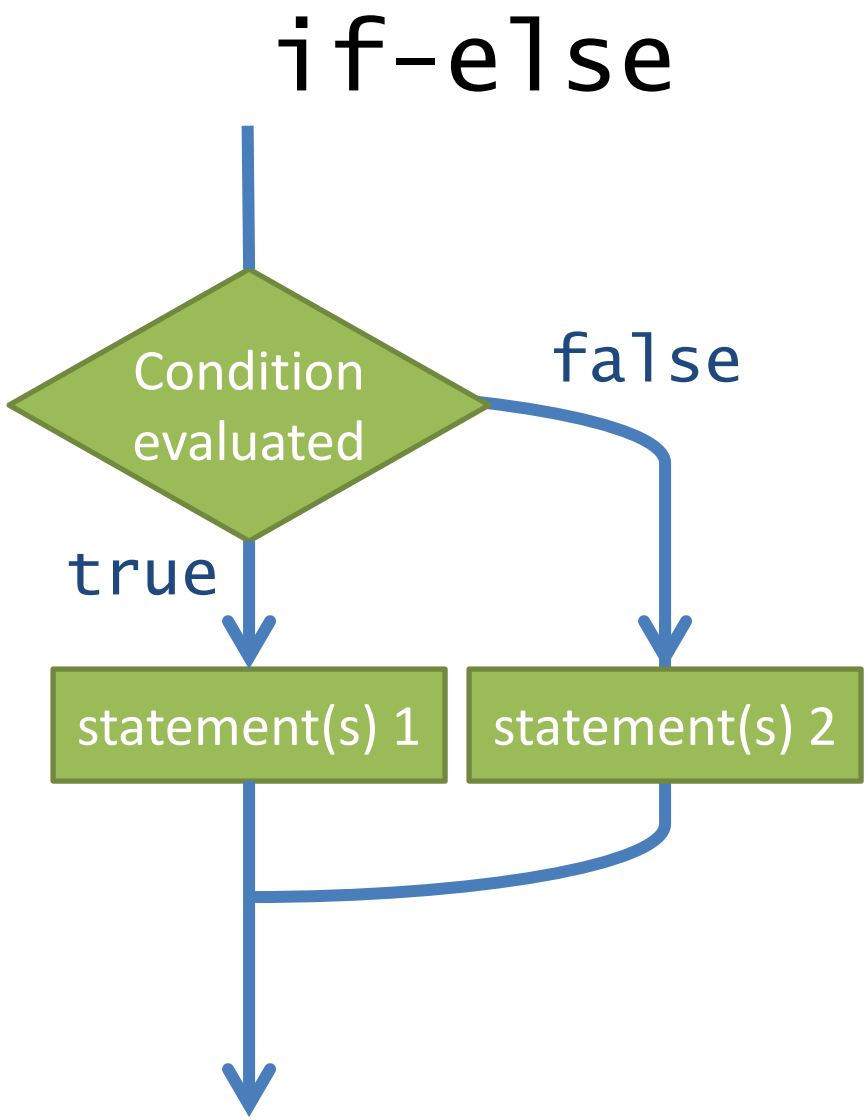


08 Making Decisions

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

Two-way branching





if and if-else syntax

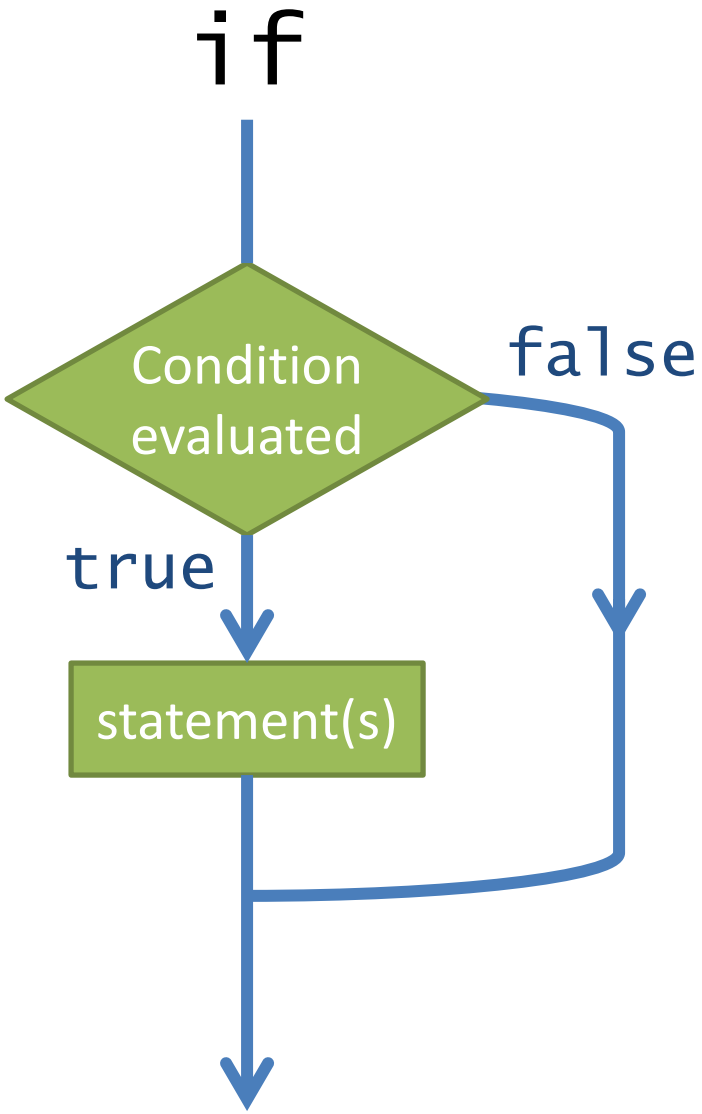
if statement

```
if ( boolean expression ) {
   statements
}
```

if-else statement

```
if ( boolean expression ) {
   statements
} else {
   statements
}
```

if on its own




```
if ( boolean expression ) {
    statements
}
```

Can they ride a rollercoaster?

Task: Decide if someone can ride the *Superman Escape* rollercoaster at WB Movie World

Knowledge: They must be between 140cm and 196cm tall and in good health (these are not *quite* the actual rules, but it's just an example)

Available data:

```
boolean hasHeartCondition;
int height;
```

Can they ride a rollercoaster?

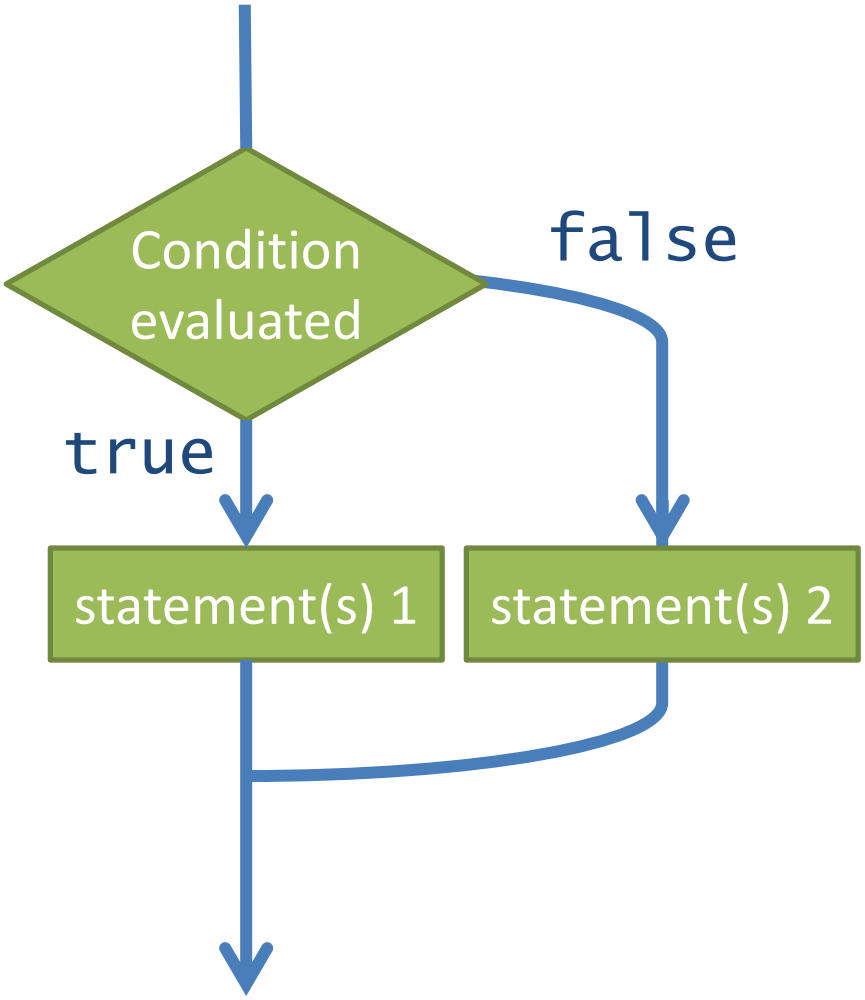
```
height >= 140 && height <= 196 && !hasHeartCondition
```

At least 140cm tall *AND no more than 196cm tall* *AND in good health*

```
if (height >= 140 && height <= 196
                          && !hasHeartCondition)
{
    //Tell them they can ride the rollercoaster
}
```

This or *that*

if-else




```
if ( boolean expression ) {
    statements 1
} else {
    statements 2
}
```

Walk or drive

Task: Decide if will walk or drive to work

Knowledge: Don't like walking in the rain

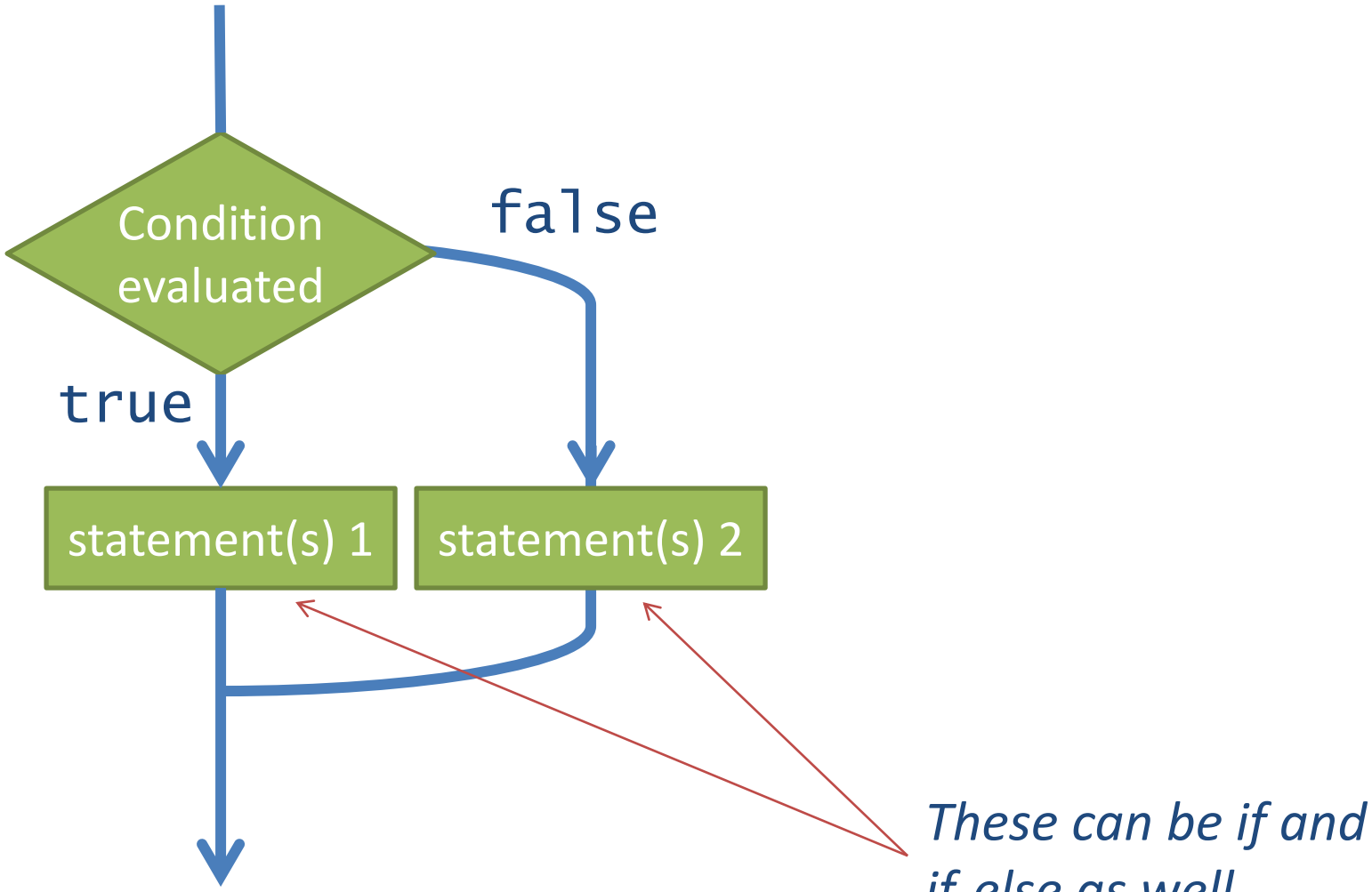
Available data:

```
boolean isRaining;
```

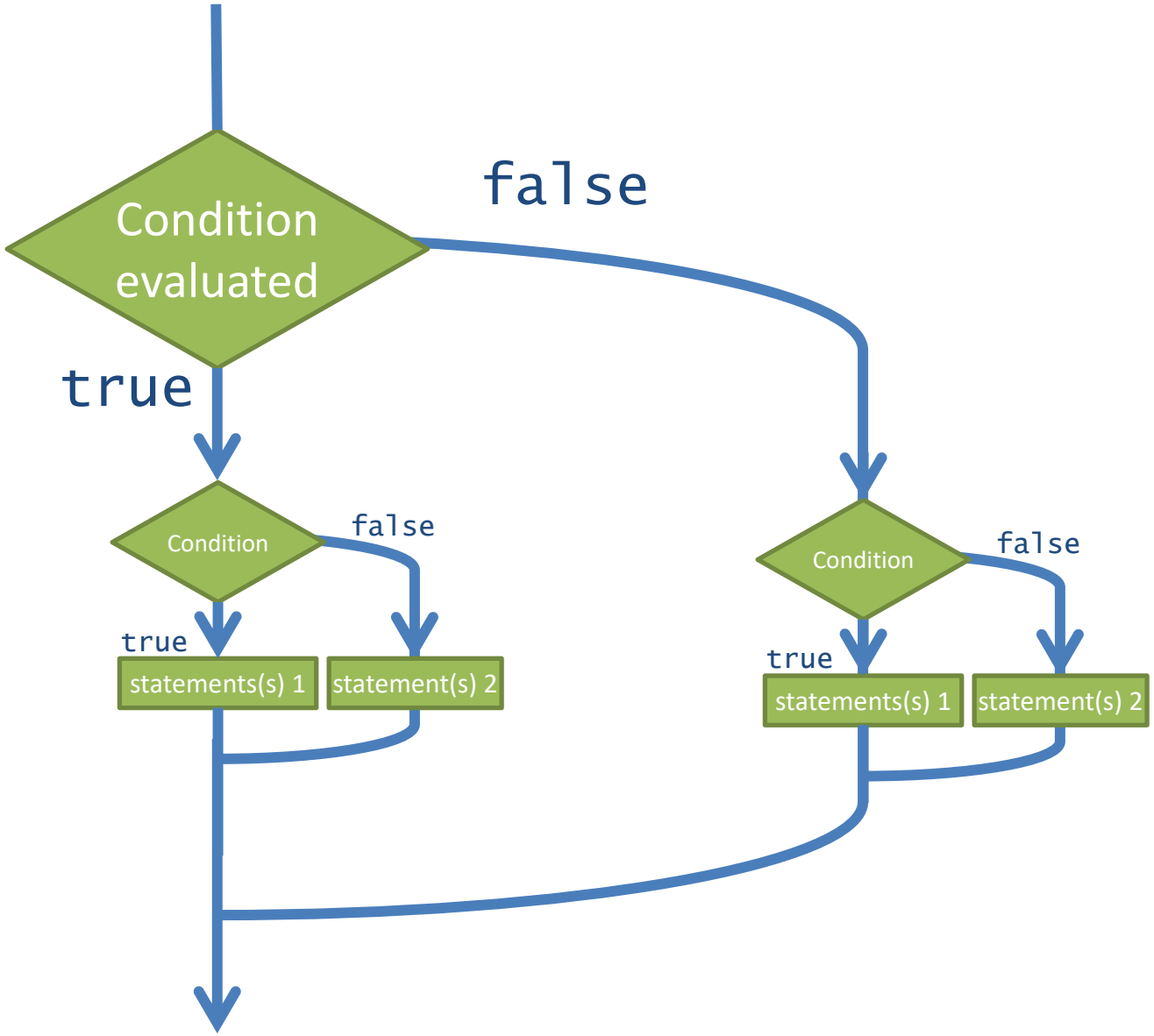
Solution:

```
if (isRaining) {
    //drive
} else {
    //walk
}
```

Nesting ifs and elses



Nesting ifs and elses



Finding the smallest of three numbers

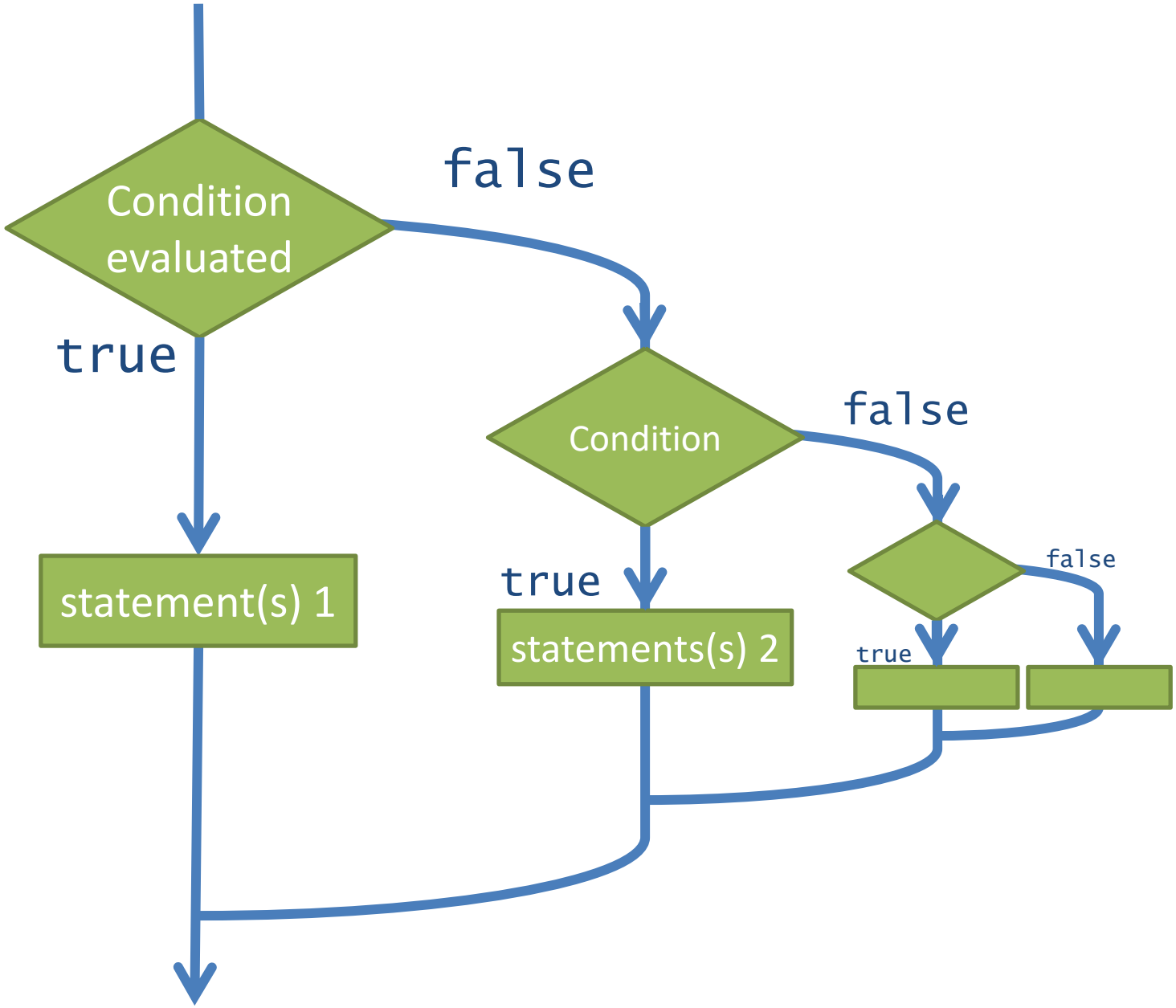
```
//Assume n1, n2 and n3 already declared and assigned
if (n1 < n2) {
    if (n1 < n3) {
        //n1 is smallest
    } else {
        //n3 is smallest
    }
} else {
    if (n2 < n3) {
        //n2 is smallest
    } else {
        //n3 is smallest
    }
}
```

Breaks up the logic into smaller tests

In the first major branch we know the n2 cannot be the smallest

see SmallestOf3.java

Sequences of ifs

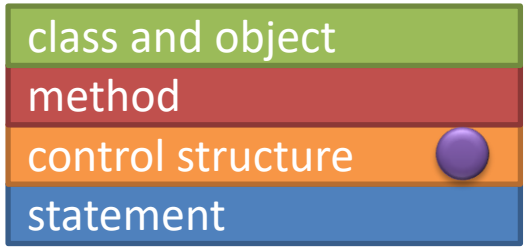


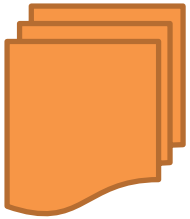
Sequences of ifs: what grade?

```
String grade;
int score; //assigned some value before...

if (score >= 80) {
    grade = "HD";
} else if (score >= 70) {
    grade = "DN";
} else if (score >= 60) {
    grade = "CR";
} else if (score >= 50) {
    grade = "PP";
} else {
    grade = "NN";
}
```

Making Decisions

Part 3: Multi-way branching with **switch**



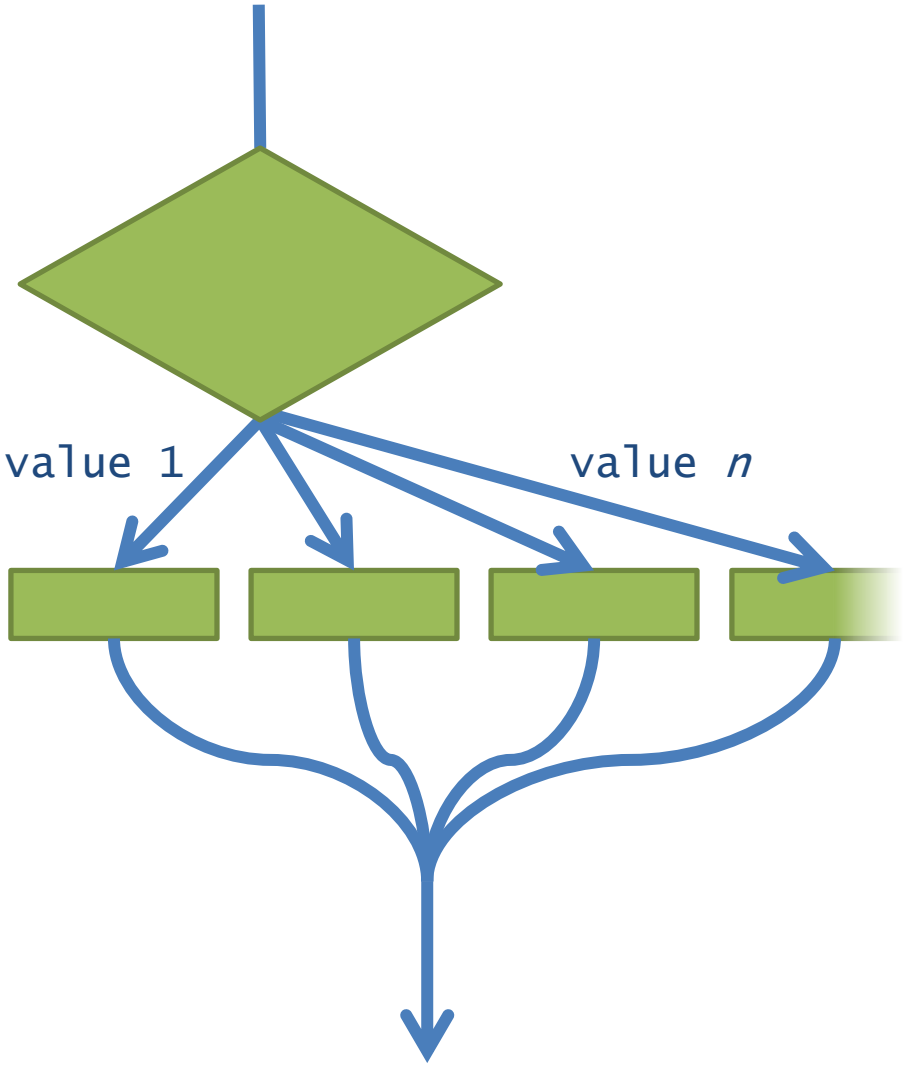


08 Making Decisions

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

Switching between alternatives

switch




```
switch ( expression ) {
    case value : statements
                 break;
    case value : statements
                 break;
    . . .
    default: statements
}
```

Switching between alternatives

```
switch  ( expression )  {
    case value : statements
                 break;
    case value : statements
                 break;

    . . .

    default: statements
}
```

The data type of *expression* can be:

int (or byte, short, long)

char

String (Java silently calls .equals() to ensure *value* equality)

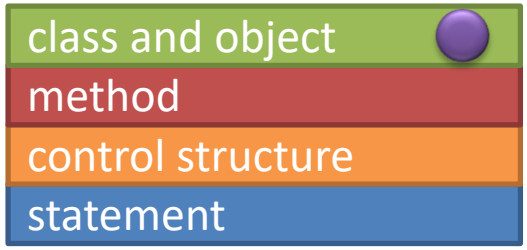
Enumerated types (next section)

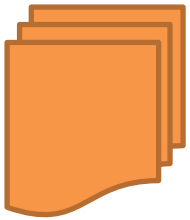
A simple menu-driven application

```
import java.util.Scanner;
public class SwitchExample {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String choice; //user's selection
        double cost; //item's price

        System.out.println("Fruit Price Check. Choices are:");
        System.out.println("(a)pple, (b)anana, (c)herry");
        System.out.print("Your selection (a-c): ");
        //Read next 1 character long 'word' from user
        choice = sc.next(); //this is _not_ a char, but a String

        switch (choice) {
            case "a": cost = 1.2; break;
            case "b": cost = 2.45; break;
            case "c": cost = 5.99; break;
            default: System.err.println("Unknown option!");
                     cost = 0;
        }
        System.out.println("Price is $" + cost);
    }
}
```

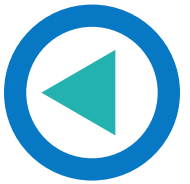
Enumerated data types

Defining your own new (simple) data types





08 Making Decisions



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

The need for enumerated types

Sometimes we need a variable to represent a very limited set of possible values. For example:

- **Game difficulty:** beginner, intermediate, expert
- **Cardinal compass direction:** N, S, E, W
- **Day of the Week:** Sunday, Monday, … Saturday
- **Student status:** Pre-enrolment, Enrolled, Studying, Graduated

But what data type to use? Are these:

- integers?
- real numbers?
- characters?
- strings?

You *could* use integers, characters or Strings

You *would* make them *constants*, since they are fixed values

enumerated types

Some constants naturally form groups (e.g., days of the week)

Sometimes *identity* matters more than *value*

We could do this…

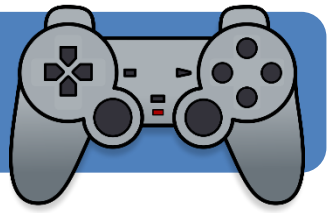
```
final int SUNDAY = 0;
final int MONDAY = 1;
final int TUESDAY = 2;
final int WEDNESDAY = 3;
final int THURSDAY = 4;
final int FRIDAY = 5;
final int SATURDAY = 6;

//Example usage:
int day = MONDAY; //OK
day = 1024; //A problem!
```

…but this is better

```
enum Day {
    SUNDAY, MONDAY,
    TUESDAY, WEDNESDAY,
    THURSDAY, FRIDAY,
    SATURDAY
}

//Example usage:
Day day = Day.MONDAY;
```

Other examples

Game difficulty



```
enum Difficulty { BEGINNER, INTERMEDIATE, EXPERT };
```

Cardinal compass direction


```
enum Direction { NORTH, SOUTH, EAST, WEST };
```

Student status

```
enum Status { PRE_ENROLMENT, ENROLLED, STUDYING,
              GRADUATED };
```

Declare enums inside class, outside methods

'public' is optional, but in later (bigger) programs will be necessary

```
public class EnumExample {

    public enum Difficulty {
        BEGINNER, INTERMEDIATE, EXPERT
    };

    public static void main(String[] args) {
        Difficulty diff = Difficulty.BEGINNER;

        //Lots of code that allows diff
        //to be changed and that makes
        //decisions based on its value

        if (diff == Difficulty.EXPERT) {
            System.out.println("Good luck!");
        }
    }
}
```

Determine day name

```
//Given Day enum defined earlier
Day today = Day.FRIDAY; //would be set elsewhere
String dayName;
switch (today) {
    case SUNDAY: dayName = "Sunday";
                 break;
    case MONDAY: dayName = "Monday";
                 break;
    case TUESDAY: dayName = "Tuesday";
                  break;
    . . .

    default: dayName = "Saturday";
}
```

see DayName.java