

Class definitions

The Ticket Machine example

Review

Last lecture we covered:

- Objects and classes
- Methods
(with parameters & return values)
- Fields and state
- Data types

Application

We have a new application: to design a simple ticket machine.

Requirements:

- All tickets cost the same amount: \$1.50
- The user adds money incrementally.
- When enough money is added, the user selects a destination and a ticket is dispensed.

Design

We need to answer some questions:

- What are the **objects** and **classes**?
- What **fields** will they have?
- What **methods** will they have?

Objects and classes

We will have two classes:

- **TicketMachine** to represent the machine itself
- **Ticket** to represent the tickets dispensed.

We will also need to represent money but we will just use an `int` for this and not create a dedicated class.

Style: Class Naming

A class should always have a **meaningful name** to describe the kind of object it represents.

We generally name classes as **nouns** (things).

Classes start with a **capital letter**.

If the name has multiple words, we used **BumpyCaps** to string them together.

TicketMachine

What **actions** can we perform on the **TicketMachine**?

- Add some money.
- Ask it to dispense a ticket.
- Ask for money to be returned.
- Ask for the ticket price.
- Ask how much money has been added so far.

TicketMachine

What **state** should the **TicketMachine** track?

- The price of the ticket.
- The amount of credit
(money added and not spent)
- The total number of tickets dispensed?
- The total amount of money collected?

Ticket

What **actions** can we perform on a **Ticket**?

- Ask what the destination is.

What **state** should a **Ticket** track?

- What its destination is.

Defining a class

To define a class in Java source we use the following syntax:

```
public class TicketMachine
{
    // class body goes here
}
```

Defining a class

access
modifier

When we define a class in Java source we use the following syntax:

```
public class TicketMachine
{
    // class body goes here
}
```

Defining a class

access
modifier

class
keyword

When defining a class in a Java source file we use the following syntax:

```
public class TicketMachine  
{  
    // class body goes here  
}
```

Defining a class

access
modifier

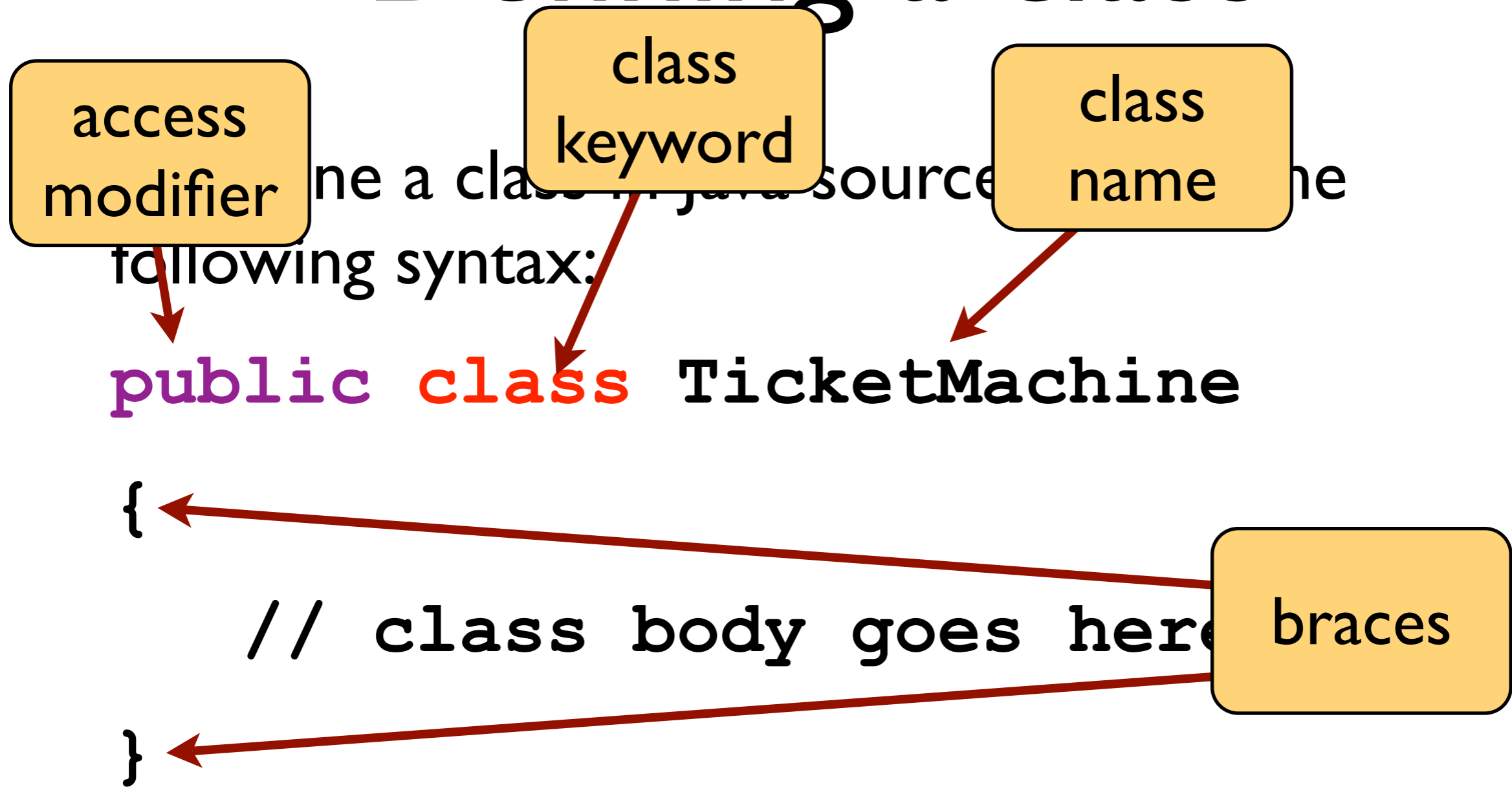
class
keyword

class
name

ne a class in java source
following syntax:

```
public class TicketMachine  
{  
    // class body goes here  
}
```

Defining a class



Parts of a class

A **class definition** has three main part:

- The **fields** of the class which store the data describing its state.
- The **constructors** which allow new objects to be created.
- The **methods** which allow us to access the state and change it.

We usually list them in this order.

Fields

We shall define three fields:

```
private int price;
```

```
private int balance;
```

```
private int total;
```

All three fields are whole numbers so we can represent them as type `int`.

Fields

access
modifier



We shall define three fields:

```
private int price;
```

```
private int balance;
```

```
private int total;
```

All three fields are whole numbers so we can represent them as type `int`.

Fields

access
modifier

type

We shall define three fields:

`private int price;`

`private int balance;`

`private int total;`

All three fields are whole numbers so we can represent them as type `int`.

Fields

access
modifier

type

name

We shall define three fields:

`private int price;`

`private int balance;`

`private int total;`

All three fields are whole numbers so we can represent them as type `int`.

Fields

access
modifier

type

name

We shall define three fields:

`private int price;`

`private int balance;`

`private int total;`

semicolon

All three fields are whole numbers so we can represent them as type `int`.

Fields are variables

Fields store pieces of information that can **vary** over time (eg: the total money in the machine).

For this reason they are called **variables**.

We will encounter other kinds of variables as we go.

Style: variables

Field names (and all other variables) should generally be **nouns** describing the property they are tracking.

Choose **meaningful** names.

By convention, Java variable start with a **lowercase** letter and use **bumpyCaps**.

Access modifiers

We have seen the keywords `public` and `private` used in our code.

These are `access modifiers` and control who can access parts of our code.

We will discuss them more later but for now you should make all `classes` and `methods` public and all `fields` private.

Comments

Java provides two ways for us to add arbitrary comments in our code:

```
// single line comment
```

```
/**  
 * Multi line  
 * comment.  
 */
```

Comments

Comments are for **documentation** and have no effect on the execution.

Good commenting **explains** the code, rather than repeating it.

Good comments

```
/**  
 * A machine which accepts money  
 * and dispenses tickets.  
 *  
 * @author Malcolm Ryan  
 * @version 1.0 (25 July 2011)  
 */  
  
public class TicketMachine ...
```

Bad comments

```
/**  
 * My TicketMachine class  
 */  
public class TicketMachine ...
```

Good comments

```
public class TicketMachine
{
    // the ticket price in cents
    private int price;

    // the total money collected
    // in cents
    private int total;
}
```

Bad comments

```
public class TicketMachine
{
    // price
    private int price;

    // total money
    private int total;
}
```

Constructors

Constructors are used to **create** new objects from a class.

It is the constructor's job to **initialise** the state of the object, i.e. to set the fields to sensible **starting values**.

`TicketMachine` constructor

The `TicketMachine` constructor needs to initialise the fields `price`, `balance` and `total`.

- The `balance` and `total` are initially zero.
- The `price` is provided as a parameter.

See `TicketMachine` source code.

Constructor syntax

The syntax of a constructor:

```
public ClassName (  
    int parameter1,  
    int parameter2)  
  
{  
    // body of the constructor  
}
```

Constructor syntax

access
modifier

The syntax of a constructor:

```
public ClassName (  
    int parameter1,  
    int parameter2)  
  
{  
    // body of the constructor  
}
```

Constructor syntax

access
modifier

same name
as class

The syntax of a constructor:

```
public ClassName (  
    int parameter1,  
    int parameter2)  
  
{  
    // body of the constructor  
}
```

Constructor syntax

access
modifier

same name
as class

parameters
with types

The syntax of a constructor:

```
public ClassName (  
    int parameter1,  
    int parameter2)  
  
{  
    // body of the constructor  
}
```

Constructor syntax

access
modifier

same name
as class

parameters
with types

The syntax of a constructor:

```
public ClassName (  
    int parameter1,  
    int parameter2)
```

```
{
```

```
// body of the constructor
```

```
}
```

braces

Parameters

We saw last week how **parameters** can be used to pass information to a method or constructor.

Parameters are another kind of **variable**. They store value values which can change.

Like fields, parameters must be defined with a **type** which describes what kind of data they contain.

Assignment

The most basic (and most common) operation in any program is to **assign a value** to a variable.

The syntax of an **assignment statement** is:

```
variable = value;
```

Eg:

```
balance = 0;
```

Assignment

We can assign values to variables many times:

```
price = 100;
```

```
// price is now 100
```

```
price = 200;
```

```
// price is now 200
```

Assignment

We can also use assignment to copy data from one variable to another:

```
balance = 100;
```

```
price = balance;
```

```
// price is now 100
```

Assignment

One important rule about assignment:

The **type of the value** on the right-hand side
must be the same as
the **type of the variable** on the left-hand side.

(Later we will discover this is not quite the whole truth...)

TicketMachine

Methods

We will need five methods for our `TicketMachine` class:

```
addMoney(int amount)
dispenseTicket(
    String destination)
refundMoney()
getPrice()
getBalance()
```

Method syntax

The syntax of a method:

```
public String doSomething(  
    int parameter1,  
    int parameter2)  
  
{  
  
    // method body  
  
}
```

Method syntax

access
modifier

The syntax of a method:

```
public String doSomething(  
    int parameter1,  
    int parameter2)  
  
{  
    // method body  
}
```

Method syntax

access
modifier

return
type

The syntax of a method:

```
public String doSomething(  
    int parameter1,  
    int parameter2)  
  
{  
    // method body  
}
```

Method syntax

access
modifier

return
type

name

The syntax of a method:

```
public String doSomething(  
    int parameter1,  
    int parameter2)  
  
{  
    // method body  
}
```

Method syntax

access
modifier

return
type

name

The syntax of a method:

```
public String doSomething(  
    int parameter1,  
    int parameter2)
```

parameters
with types

```
{
```

```
    // method body
```

```
}
```

Method syntax

access
modifier

return
type

name

The syntax of a method:

```
public String doSomething(  
    int parameter1,  
    int parameter2)
```

parameters
with types

```
{
```

```
    // method body
```

```
}
```

Method syntax

access
modifier

return
type

name

The syntax of a method:

```
public String doSomething(  
    int parameter1,  
    int parameter2)
```

parameters
with types

```
{  
    // method body  
}
```

braces

Methods vs Constructors

The headers of methods and constructors are very similar with two important differences:

```
public TicketMachine ()
```

```
public int getPrice ()
```

```
public void addMoney (int amount)
```

Methods vs

Constructors

The constructor name is **class name.** Constructors are very similar with two important differences:

```
public TicketMachine ()
```

```
public int getPrice ()
```

```
public void addMoney (int amount)
```

Methods vs

Constructors

The constructor name is **class name.** Constructors are very similar with two important differences:

```
public TicketMachine ()
```

methods can have **any** name

```
public int getPrice ()
```

```
public void addMoney (int amount)
```

Methods vs Constructors

The headers of methods and constructors are very similar with two important differences:

```
public TicketMachine ()
```

```
public int getPrice ()
```

```
public void addMoney (int amount)
```

Methods vs

Constructors

The constructors are very similar with two important differences:

constructor has no **return type**

`public TicketMachine ()`

`public int getPrice ()`

`public void addMoney (int amount)`

Methods vs

Constructors

The **constructor** has no **return type**. Constructors are very similar with two important differences:

```
public TicketMachine ()
```

methods **must** have a return type (possibly `void`)

```
public int getPrice ()
```

```
public void addMoney (int amount)
```

Method bodies

A method's body is a **sequence of statements** terminated by **semi-colons (;)**.

Each statement is executed in order.

So far we have encountered only one kind of statement, assignment.

We will encounter more as we go.

Expressions

We can use an **expression** to calculate a new value out of existing values or variables and assign it to a variable:

```
price = 50 + 25;
```

```
// price is now 75
```

```
price = price * 2
```

```
// price is now 150
```

Expressions

Assignment statements of the form:

```
balance = balance + money;
```

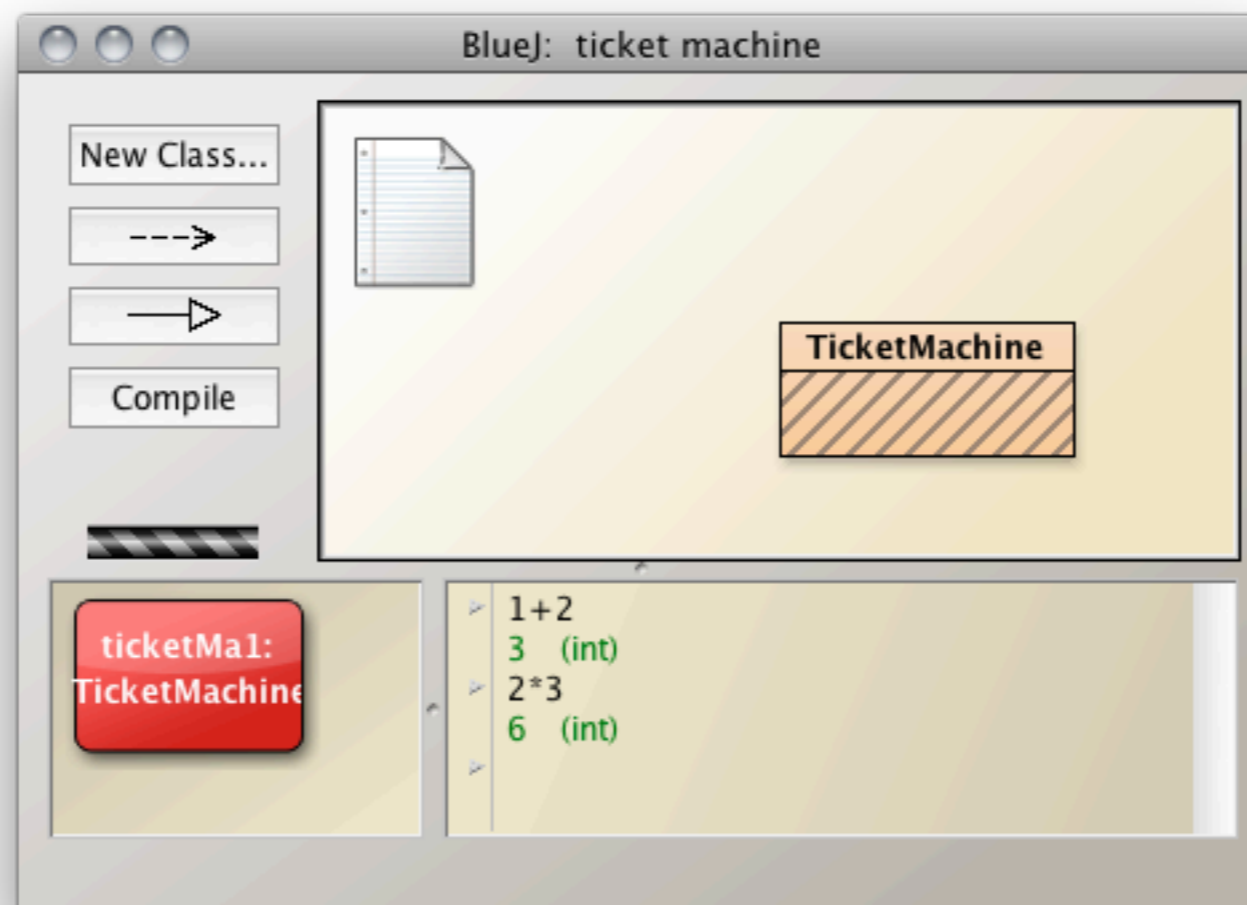
Are so common Java provides a shorter form:

```
balance += money;
```

There are also versions for **-**, ***** and **/**.

Code Pad

BlueJ has a Code Pad to allow you to experiment with expressions.



Style note: braces

Java uses **braces** { } (aka curly brackets) to divide code into **blocks**.

There are many common styles for braces. The BlueJ Style Guide recommends placing braces **on a new line alone**.

Choosing a standard and sticking to it is the most important thing for readable code.

Style note: indenting

Whatever style of bracketing you use, it is good style to **indent** all the code inside the braces (i.e. every block).

Indenting should be **consistent**. Usually a single tab is used.

Whitespace usually does not effect the **correctness** of your program but it can have a major effect on its **readability**.

Using the Debugger

BlueJ provides a **debugger** to allow us to watch our code as it executes.

To use the activate debugger we first need to set a **breakpoint** in our code.

When BlueJ gets to the breakpoint it will **stop** and display the code and the debug window.

We can now **step** through line-by-line.

Kinds of Methods

There are two basic kinds of methods:

- **Accessor methods** which **access** information about an object's state.
- **Mutator methods** which **change** an object's state.

Java doesn't make a formal difference between the two but it is a useful idea to distinguish them in your code.

Returning values

Accessor methods return a value.

They must therefore have a non-void **return type**.

```
public int getPrice()
```

```
public String getDestination()
```

Return statements

To return a value we use the **return statement**:

```
return value;
```

The type of the value returned must match the return type in the header.

The return statement must be the **last** statement of the method. Statements after it will not be executed.

Variables and Scope

Every variable has a fixed lifetime, called its **scope**.

A field variable exists for the **lifetime of its object**.

A parameter variable only exists **until its method finishes**.

At the end of its scope a variable is **forgotten**.

Local variables

A **local variable** is a variable whose scope is only one method.

Local variables are useful as **temporary space** for performing calculations.

A local variable should always be **initialised**.

Local variables

The syntax for a local variable is:

```
public void method()  
{  
    int localVariable =  
                                initialValue;  
  
    // inside scope  
}  
  
// outside of scope
```

Fields, parameters & local variables

All three kinds of variables:

- store data
- have a type
- have a fixed lifetime

Fields, parameters & local variables

Fields:

- are defined **outside** of methods.
- are initialised in the **constructor**.
- exist until their object is **destroyed**.

Fields, parameters & local variables

Parameters:

- are defined in the **method header**.
- are initialised by the **caller**.
- exist until the method **ends**.

Fields, parameters & local variables

Local variables:

- are defined in the **method body**.
- are initialised **when defined**.
- exist until the **method ends**.

Conditional statements

The syntax of a conditional statement:

```
if (condition) {  
    // do this if true  
}  
else {  
    // do this if false  
}  
  
// continue here in either case
```

Conditional statements

Sometimes we need our code to do different things depending on its input.

We use a **conditional statement** (**if statement**) to perform a test and branch based on its results.

Conditional statements

The 'else' part of a conditional is optional:

```
if (condition) {  
    // do this if true  
}  
// continue here in either case
```

Boolean expressions

The condition of an if-statement is a **boolean** value.

A boolean is a data type that can have only two values **true** or **false**.

Boolean expressions

Example boolean expressions:

```
x == y // equality
```

```
x != y // inequality
```

```
x < y // less than
```

```
x <= y // less or equals
```

Note that we use `==` for **equality**
and `=` for **assignment**.