CT2106
Object Oriented Programming

Dr. Frank Glavin
Room 404, IT Building
Frank.Glavin@University*of*Galway.ie
School of Computer Science

University *of* Galway.ie

Last Lecture - First Java Code

- In the last session, you wrote your first class and created several objects from it
- You were introduced to the notion of **state**
  - **Every object has its own state**
- An object's state is generally defined by the values it holds
- Multiple objects can be created from a single class. Each object can have its own state.

Ollscoil na Gaillimhe
University of Galway

Topics

By the end of this lecture you will be able to implement the following in Java:

- Correct class and method structure
- Define and initialise an int variable
- Use accessor and mutator methods
- Explain the concept of encapsulation
- Print out the object state
- Use the Java conditional statement (if else)

Ollscoil na Gaillimhe
University of Galway

Today's Learning exercise

In Blue J:

- Create a Bicycle class and a Car class
- Each Bicycle object should its own speed, gear and cadence (e.g. 1st, 2nd, 3rd etc) state
- What type of variable in Java could be used to represent speed, gear and cadence (look it up on the Web)?
- Create **setSpeed**, **setGear** and **setCadence** method that can set the speed /gear state of a bicycle and a car object **and print out the current speed of each**
- Then Create 3 Bicycle and 3 Car objects
- Using the methods above set and print different speed, gear and cadence values for each

Ollscoil na Gaillimhe
University of Galway

Class Structure:

Every class has the following structure

```
public class ClassName
{
    Fields
    Constructors
    Methods
}
```

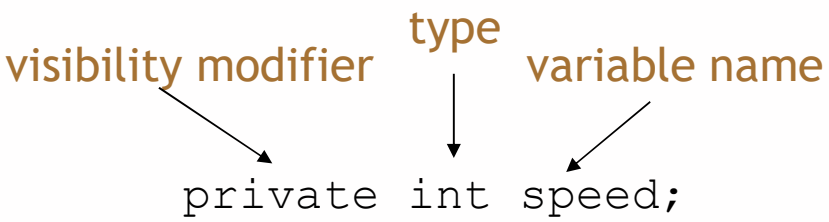
Ollscoil na Gaillimhe
University of Galway

Fields

- Fields store values for an object.
- They are also known as *instance variables*.
- Fields define the state of an object.
- Use *Inspect* in BlueJ to view the state.
- Some values change often.
- Some change rarely (or not at all).

```
public class Bicycle
{
    private int speed;
    private int gear;
    private int cadence;

    Further details omitted.
}
```



Data Type: int

Java Primitive Types

Type	Size	Range	Default
boolean	1 bit	true or false	false
byte	8 bits	[-128, 127]	0
short	16 bits	[-32,768, 32,767]	0
char	16 bits	['\u0000', '\uffff'] or [0, 65535]	'\u0000'
int	32 bits	[-2,147,483,648 to 2,147,483,647]	0
long	64 bits	$[-2^{63}, 2^{63}-1]$	0
float	32 bits	32-bit IEEE 754 floating-point	0.0
double	64 bits	64-bit IEEE 754 floating-point	0.0

Ollscoil na Gaillimhe
University of Galway

Principle 1 of OOP: Encapsulation

In encapsulation, the variables of a class will be hidden from other classes and can be accessed only through the methods of their current class, therefore it is also known as **data hiding**.

- Why?
- Basic OOP philosophy**: each object is responsible for its own data**
- This allows an object to have much greater **control**
  - Which data is available to be viewed externally
  - How external objects may change (mutate) the object's state

Ollscoil na Gaillimhe
University of Galway

Encapsulation Type: Private

- Making the fields **private** encapsulates their values inside each object
- No external class or object can access them.

```
public class Bicycle
{
    private int speed;
    private int gear;
    private int cadence;

    Further details omitted.
}
```

Ollscoil na Gaillimhe
University of Galway

Constructors (1)

- Initialize an object.
- Have the same name as their class.
- Close association with the fields:
  - Initial values stored into the fields.
  - Parameter values often used for these.

```
public Bicycle(int spd, int gr, int cad)
{
    speed = spd;
    gear = gr;
    cadence = cad;
}
```

Ollscoil na Gaillimhe
University of Galway

Constructors (2)

- If input parameter variables have the **same name** as your fields
- Then you must use the **this** keyword to distinguish between the two
- this = “belonging to this object”

```
public Bicycle(int speed, int gear, int cadence)
{
    this.speed = speed;
    this.gear = gear;
    this.cadence = cadence;
}
```

Ollscoil na Gaillimhe
University of Galway

Choosing Variable Names

- There is a lot of freedom over choice of names. Use it wisely!
- Choose expressive names to make code easier to understand:
  - price, amount, name, age, etc.
- Avoid single-letter or cryptic names:
  - w, t5, xyz123

Ollscoil na Gaillimhe
University of Galway

Methods

- Methods implement the *behaviour* of an object.
- Methods have a consistent structure comprised of a *header* and a *body*.

- ***Accessor** methods* provide information about the state of an object.
- ***Mutator** methods* alter the state of an object.
- Other sorts of methods accomplish a variety of tasks.

Ollscoil na Gaillimhe
University of Galway

Method structure

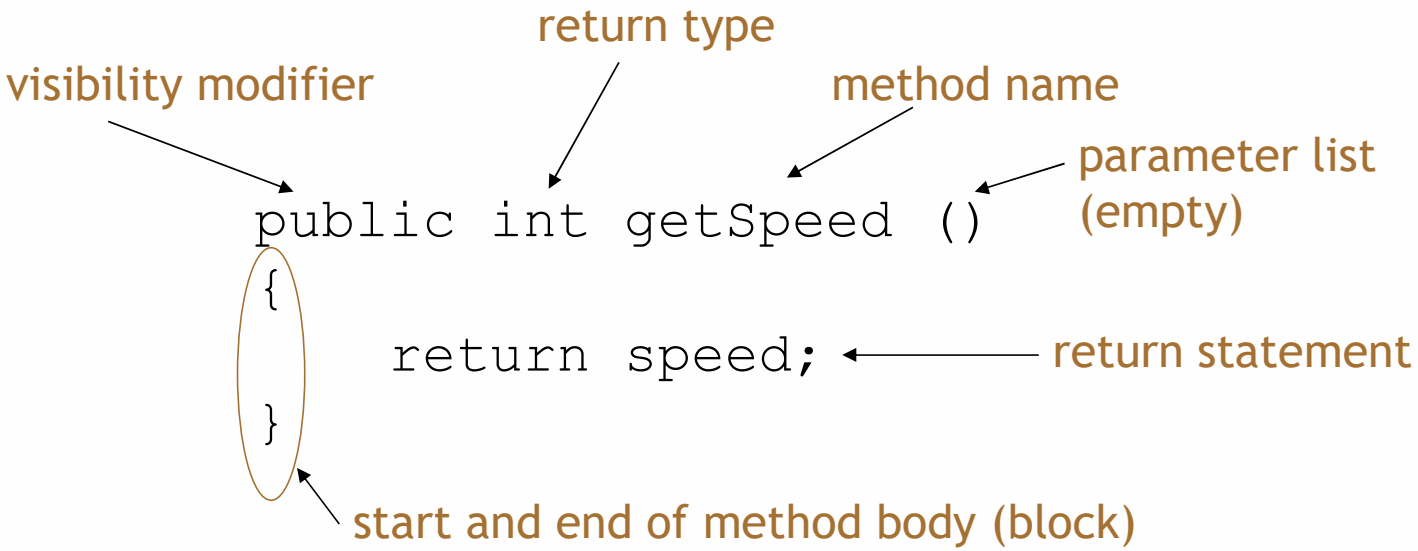
- The header:
  - **public int getSpeed ()**

- The header tells us:
  - the *visibility* to objects of other classes;
  - whether the method *returns a result*;
  - the *name* of the method;
  - whether the method takes *parameters*.

- The body encloses the method's *statements*.

Ollscoil na Gaillimhe
University of Galway

Accessor (**get**) methods



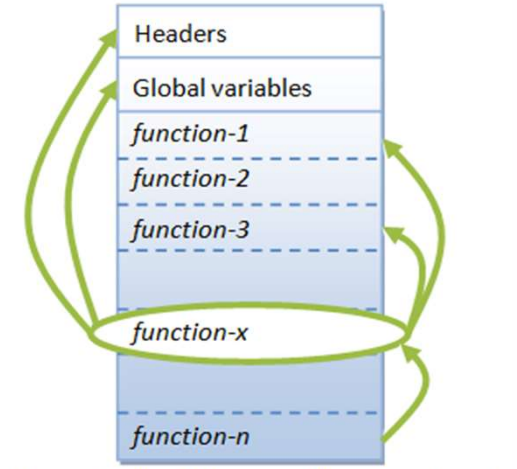
Ollscoil na Gaillimhe
University of Galway

Accessor methods

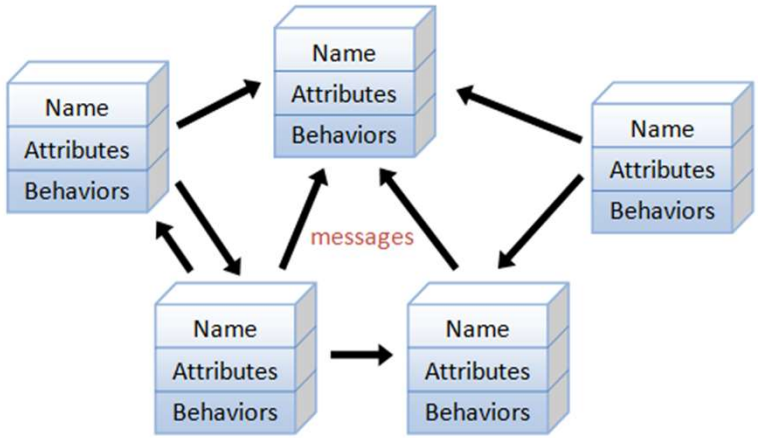
- An accessor method always has a return type that is not **void**.
- An accessor method returns a value (*result*) of the type given in the header.
- The method will contain a **return** statement to return the value.
- NB: Returning is *not* printing!

Ollscoil na Gaillimhe
University of Galway

C vs. Java



A function (in C) is not well-encapsulated



An object-oriented program consists of many well-encapsulated objects and interacting with each other by sending messages

- Unlike a C program, an OOP program **will not** have a pool of global variables that each method can access
- Instead, **each object has its own data** – and other objects rely upon the *accessor* methods of the object to access the data

Ollscoil na Gaillimhe
University of Galway

```
public class Bicycle {

    private int cadence;
    private int speed;
    private int gear;

    public int getCadence() {
        return cadence;
    }

    public void setCadence(int newValue) {
        cadence = newValue;
    }

    public int getGear() {
        return gear;
    }

    public void setGear(int newValue) {
        gear = newValue;
    }

    public int getSpeed() {
        return speed;
    }
                ...
```

- The instance variables (or fields) are declared private
- Cannot be accessed directly

- accessor/mutator methods used to access the data

- These are often called **getter/setter** methods

Test:

```
public class Bicycle
{
private speed;

public Bicycle()
{
    speed = 300
}

public int getSpeed
{
    return Speed;
}
```

What is wrong here?

(there are five errors!)

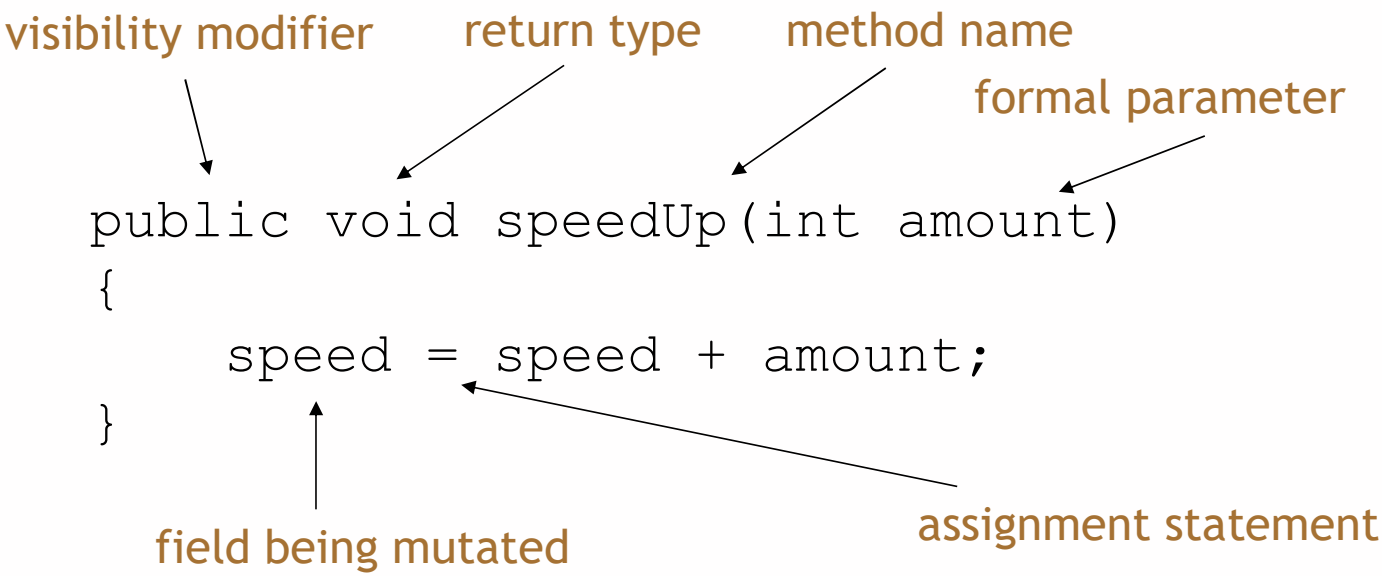
Ollscoil na Gaillimhe
University of Galway

Mutator Methods (1)

- Have a similar method structure: header and body.
- Used to *mutate* (i.e., change) an object's state.
- Achieved through changing the value of one or more fields.
  They typically contain one or more assignment statements.
  Often receive parameters.

Ollscoil na Gaillimhe
University of Galway

Mutator Methods (2)



Ollscoil na Gaillimhe
University of Galway

Mutator Methods: ‘set’

- Each field may have a dedicated **set** mutator method.

- These have a simple, distinctive form:
  void return type
  method name related to the field name
  single formal parameter, with the same type as the type of the field
  a single assignment statement

Ollscoil na Gaillimhe
University of Galway

Mutator Methods: ‘set’

- A typical ‘**set’** method

```
public void setGear(int number)
{
    gear = number;
}
```

- We can easily infer that gear is a field of type ‘int’,
  - private int gear;

Ollscoil na Gaillimhe
University of Galway

Protective Mutators

- A set method does not have to always assign unconditionally to the field.
- The parameter may be checked for validity and rejected if inappropriate.
- Mutators thereby protect fields.
- Mutators support *encapsulation*.

Ollscoil na Gaillimhe
University of Galway

Printing From Methods

```
public void printState()
{
    // Simulates output from a bike computer.
    System.out.println("####################");
    System.out.println("# Speed: " + speed);
    System.out.println("# Gear : " + gear);
    System.out.println("# Cadence: " + cadence);
    System.out.println("####################");
    System.out.println();
}
```

Ollscoil na Gaillimhe
University of Galway

Printing From Methods 2

```
public void printState()
{
    // Simulates output from a bike computer.
    System.out.println("####################");
    System.out.printf("# Speed: %d \n ", speed);
    System.out.printf("# Gear : %d \n ", gear);
    System.out.printf("# Cadence: %d \n", cadence);
    System.out.println("####################");
    System.out.println();
}
```

Ollscoil na Gaillimhe
University of Galway

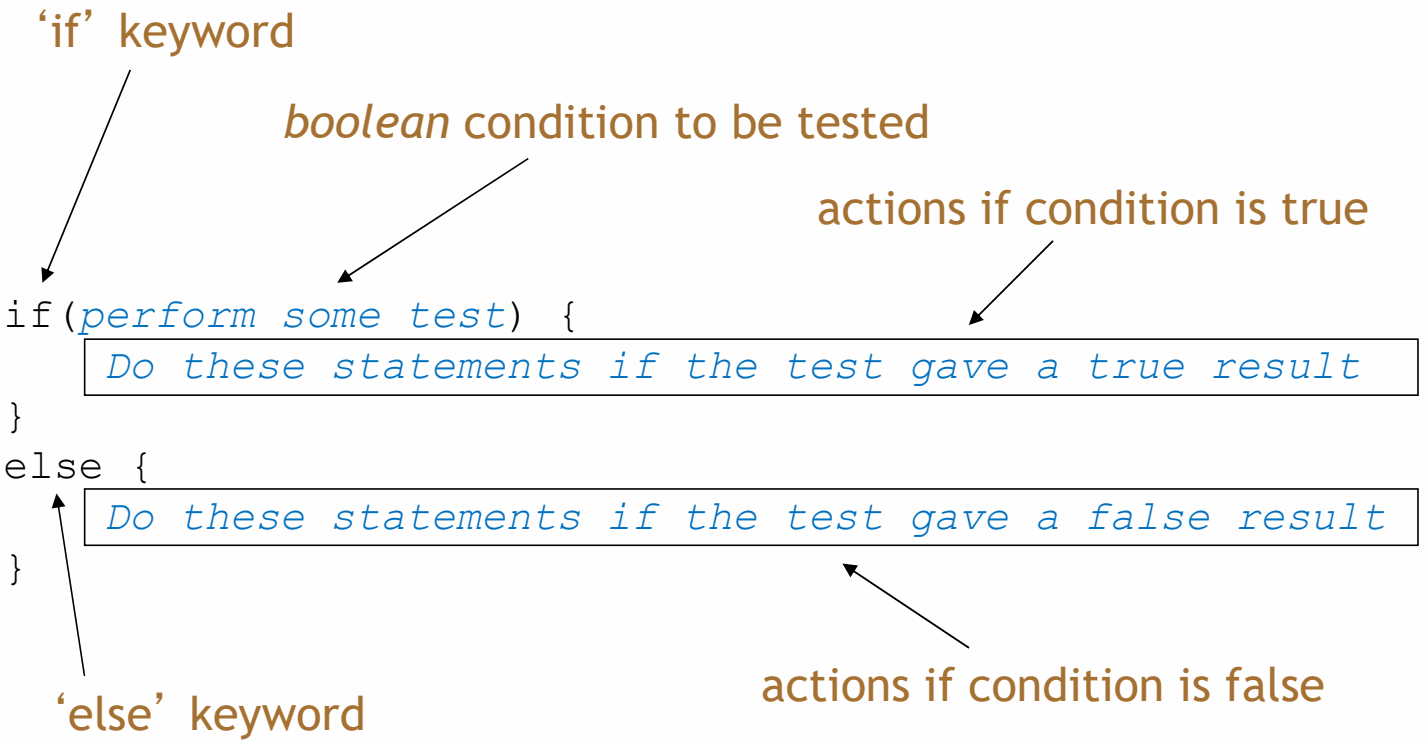
Conditional Statement

```
if(I have enough money left) {
    I will go out for a meal;
} else {
    I will stay home and watch a movie;
}
```

- It has the same format that you have seen in C

Ollscoil na Gaillimhe
University of Galway

Making choices in Java



Ollscoil na Gaillimhe
University of Galway

Protecting a Field (1)

```
public void setGear(int gearing)
{
    if(gearing <= 18) {
        gear = gearing;
    }
    else {
        System.out.println(
            ”Exceeds maximum gear ratio.
                        Gear not set”);
    }
}
```

This conditional statement avoids an inappropriate action. It protects the gear field from too large values

Ollscoil na Gaillimhe
University of Galway

Protecting a Field (2)

```
public void setGear(int gearing)
{
    if(gearing >= 1 && gearing <= 18) {
        gear = gearing;
    }
    else {
        System.out.println(
            ”gear input value not in the
                                correct range”);
    }
}
```

Ollscoil na Gaillimhe
University of Galway

This conditional statement avoids an inappropriate action. It protects the gear field from too large AND too small values

Summary

- You have encountered some of the fundamental ideas in OOP
- A class has fields, a constructor(s) and methods
- Encapsulation - each object's data (fields) is protected by its accessor/mutator methods
- If you want to access/change an object's state, you must use its accessor/mutator methods
- The use of the 'private' keyword prevents external access to an object's data

Ollscoil na Gaillimhe
University of Galway