NATURAL ENVIRONMENT RESEARCH COUNCIL

EPSRC

Introduction to Object-Oriented Programming

archer

epcc

THE UNIVERSITY OF EDINBURGH

Reusing this material

Procedural programming

- Traditional scientific programming languages (i.e. C, Fortran) are procedural
  - Programs constructed from functions/subroutines/procedures
  - Modularity and re-use of operations achieved through grouping in functions
  - Data scoping generally based on function scope
- Single function required (main), better practise to have as many functions as match distinct operations in the program
- Generally no explicit link between data and functions
- Data accessible and modifiable by functions at will

Object-oriented programming

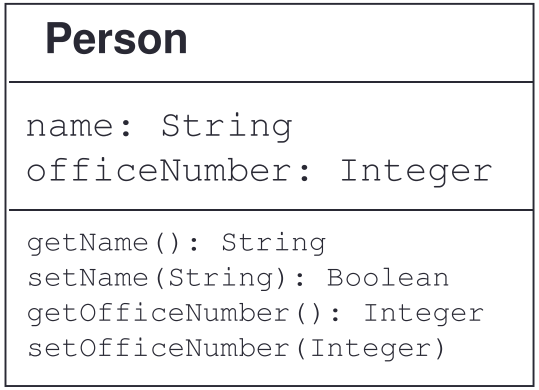
- Object-oriented programming (OOP)
  - Large programs often become hard to maintain and extend
  - Complex interdependencies makes development and maintenance difficult
  - Packaging functionality and data into groups and only exposing the minimum amount of this to other parts of the program can help address this issue
- Abstract datatypes (ADTs) are attractive programming idea
  - Group code and data together
  - Hide data and only access through associated code
  - Provide defined interfaces and access mechanisms
  - Hide users of data from details

OOP

- Can implement ADTs in procedural lanugages
  - Derived datatypes/structures
  - Doesn’t force hiding of data
  - Don’t allow easy re-use for different datatypes
- Object concept designed to allow fully functional implementation of ADTs
  - Allow better control of visibility and access
  - Allow better re-use of common code and extension for different data types or functionality

OOP - Encapsulation

- A class is a specification of an ADT
  - Blueprint of the ADT, definition of data and implementation of procedures
- An instance is a runtime instantiation of the ADT
  - Actual ADT with data in it
  - Can have as many instances as the program requires
  - Instance also known as an Object

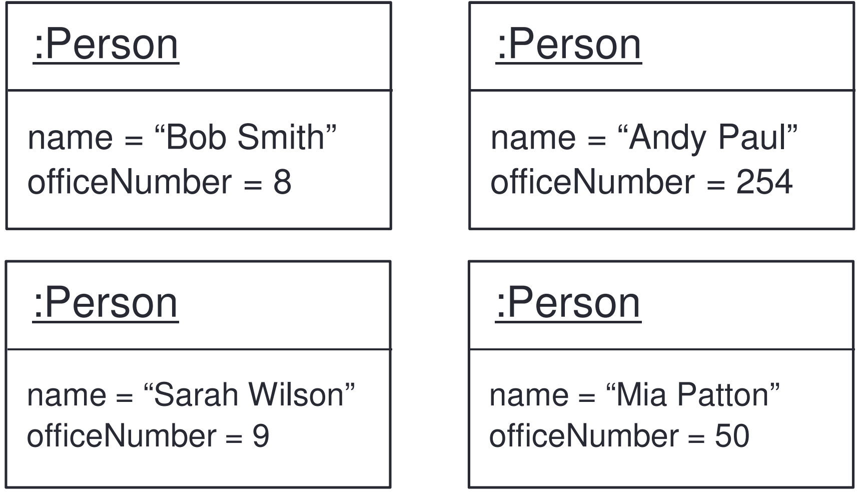


Person

```
name: String
officeNumber: Integer
```

```
getName(): String
setName(String): Boolean
getOfficeNumber(): Integer
setOfficeNumber(Integer)
```

Class



:Person

name = “Bob Smith”
officeNumber = 8

:Person

name = “Andy Paul”
officeNumber = 254

:Person

name = “Sarah Wilson”
officeNumber = 9

:Person

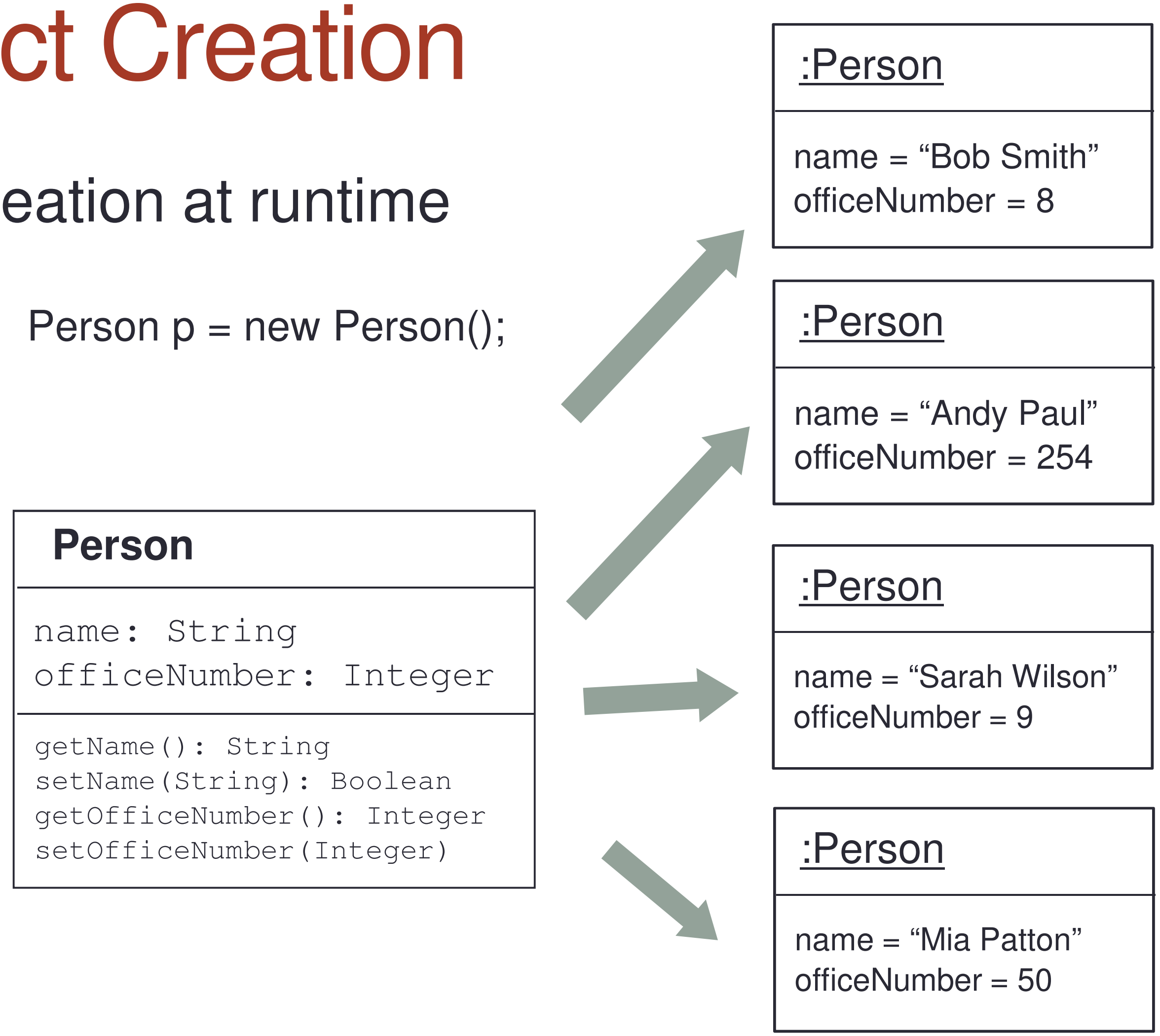
name = “Mia Patton”
officeNumber = 50

Runtime Instances/Objects

Object Creation

- Object creation at runtime

Person p = new Person();



Person

```
name: String
officeNumber: Integer

getName(): String
setName(String): Boolean
getOfficeNumber(): Integer
setOfficeNumber(Integer)
```

:Person

name = “Bob Smith”
officeNumber = 8

:Person

name = “Andy Paul”
officeNumber = 254

:Person

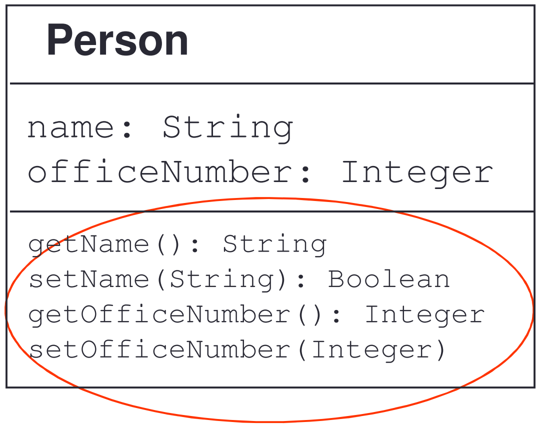
name = “Sarah Wilson”
officeNumber = 9

:Person

name = “Mia Patton”
officeNumber = 50

Methods - Encapsulation

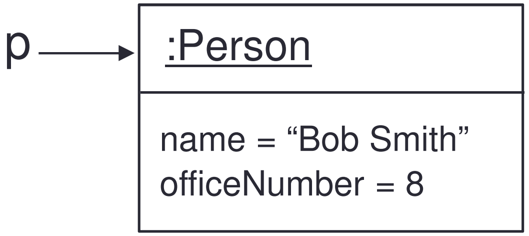
- Objects used by calling *methods*
  - methods in OO => functions/subroutines in procedural programming
  - they can take arguments and return results
  - Cannot be called in isolation
    - need an instance (object)



Person

```
name: String
officeNumber: Integer
```

```
getName(): String
setName(String): Boolean
getOfficeNumber(): Integer
setOfficeNumber(Integer)
```



p → :Person

name = “Bob Smith”
officeNumber = 8

p.setOfficeNumber(2) ✓

p.moveOffice(); ✗

setOfficeNumber(2); ✗

Methods - Encapsulation

- Method implementation specified in class
    - Classes can have method specifications but not implementations (abstract class)
    - Abstract classes need to be implemented in other classes to be used
- Methods can access class data
    - i.e. `officeNumber` not visible outside class

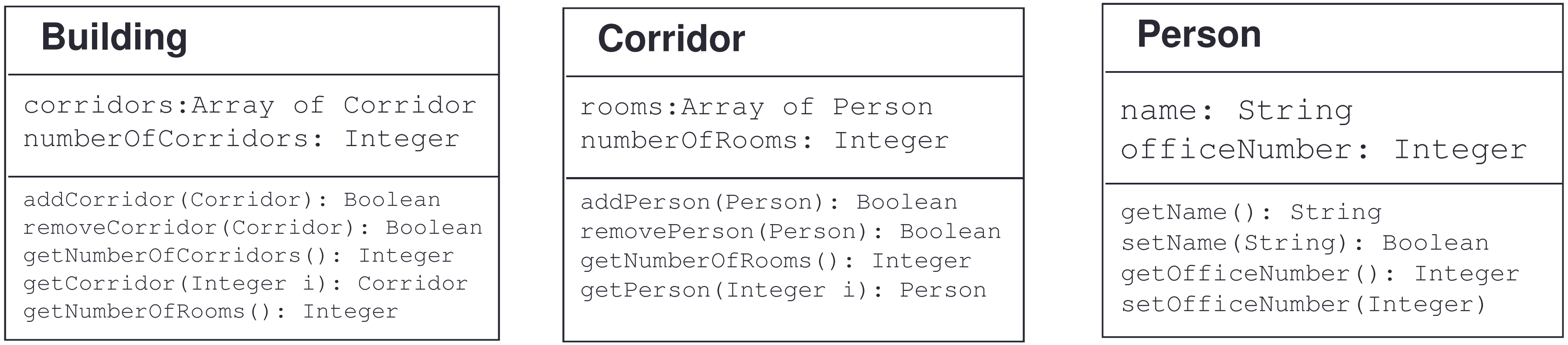
```
// Person
public void setOfficeNumber(integer number) {
  officeNumber = number;
  return;
}
```

Class hierarchy and relationships

- Individual class functionality not particularly useful
  - Power of OOP comes from the relationships between multiple classes
  - Controlling code and data re-use
  - Defining hierarchy/relationships between data
- Composition: has-a
  - Objects can contain other objects within them
- Inheritance: is-a-type-of
  - Objects can be built on other objects (extend)
  - Allows for multiple version of some functionality, only changing or adding what is required for the different version
  - Abstract classes can define object to be implemented with no actual implementation provided

Composition

- Class containing an object can use the objects methods through that object



Building

```
corridors:Array of Corridor
numberOfCorridors: Integer

addCorridor(Corridor): Boolean
removeCorridor(Corridor): Boolean
getNumberOfCorridors(): Integer
getCorridor(Integer i): Corridor
getNumberOfRooms(): Integer
```

Corridor

```
rooms:Array of Person
numberOfRooms: Integer

addPerson(Person): Boolean
removePerson(Person): Boolean
getNumberOfRooms(): Integer
getPerson(Integer i): Person
```

Person

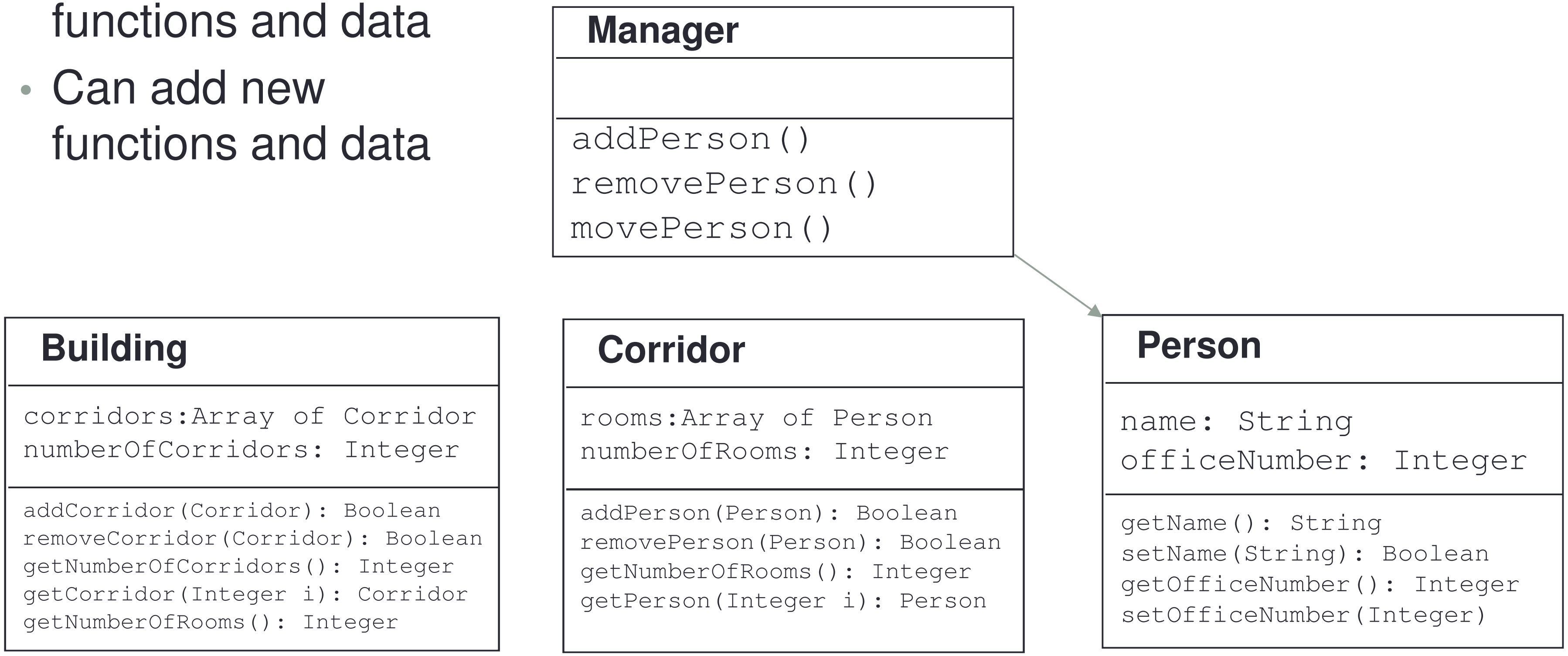
```
name: String
officeNumber: Integer

getName(): String
setName(String): Boolean
getOfficeNumber(): Integer
setOfficeNumber(Integer)
```

Inheritance

- Subclass inherits (can use) superclass functions and data
- Can add new functions and data

- Manager is a subclass of Person
- Person is a superclass of Manager



Manager

addPerson()
removePerson()
movePerson()

Building

corridors:Array of Corridor
numberOfCorridors: Integer

addCorridor(Corridor): Boolean
removeCorridor(Corridor): Boolean
getNumberOfCorridors(): Integer
getCorridor(Integer i): Corridor
getNumberOfRooms(): Integer

Corridor

rooms:Array of Person
numberOfRooms: Integer

addPerson(Person): Boolean
removePerson(Person): Boolean
getNumberOfRooms(): Integer
getPerson(Integer i): Person

Person

name: String
officeNumber: Integer

getName(): String
setName(String): Boolean
getOfficeNumber(): Integer
setOfficeNumber(Integer)

Polymorphism

- Can re-define superclass functions (override/subtyping)

```
// Person
public void print() {
  write(“Person: “, name);
  return;
}
//Manager
public void print() {
  write(“Manager: “, name);
  return;
}
```

Mixing objects

- OO languages allow mixing of subclasses:

```
Person arr[] = new Person[2];
arr[0] = new Person(“Adrian Jackson”);
arr[1] = new Manager(“David Henty”);

for (int i = 0; i < numPeople; i++) {
 arr[i].print();
}
```

Construction/Destruction

- Special function/method to setup an object (constructor)
  - Called on object creation
  - Ensures that object is created in desired state

- Likewise, possible to provide a method that is called when an object is destroyed (destructor)
  - Enable cleaning up when object is no longer needed

Summary

- Object-oriented programming groups data and functionality together
    - Safety of data can be ensured by controlling how data is accessed and updated (encapsulation)
- Definition and instance of data separated into class and object
    - Class defines the data and functions that can operate on that data
    - Object is a specific instance of the class
    - Can have many, separate and distinct, objects from the same class
- Functionality can be re-used when creating new objects
    - Composition and inheritance
- Enable specification/provision of interfaces for others to use or implement

Exercise

- This is purely a thought exercise
- Have a look at the percolate code. Think about how you could split that into classes
  - Where could different functionality go?
  - How would it then be used?
  - What needs to know/access what?
  - Where could things be likely to change?
  - What things need to hidden for safety?