Constructors

Introduction to OO Programming

Constructors

- A constructor is a method which is called when instantiating an object.

```
BankAccount myAccount = new BankAccount();
Converter c = new Converter();
Enemy ghost  = new Enemy();
```

- A constructor creates and initialises the instance variables of an object.
- The code in a constructor body will assign values to the instance fields of the object that is being constructed.

Designing classes with constructors

```
public class ClassName
{
  //instance variables
  …
  //constructors
  …
  //methods
  …
}
```

By convention constructor definitions are placed between variable declaration and method definitions

Constructor Definition

A constructor is a special type of method. It differs from other methods in 3 ways:

- A constructor runs once – when an object is initialised. It sets the instance variables to initial values.
- A constructor name **must** be same as `ClassName`
- A constructor has **no** return type

```
public BankAccount()
{
        balance = 0.0
}
```

Default Constructor

- A class can be defined with no constructor. In this case, Java will call the *default constructor*. This sets each of the instance variables to either 0, `false` or `null`.
  - Numeric instance variables are set to 0.
  - Boolean variables are set to `false`.
  - Object instance variables are set to `null`.

Default constructor

```java
public class Pen
{
  // instance variable
  private double price;
  private String colour;

  // methods
  …
  …

}
```

No constructor present so java provides a default no-argument constructor

```java
public class PenTester
{
 public static void main(String[] args)
 {

  // create a Pen object

  Pen myPen = new Pen();
  …
  …

}
```

User defined Constructor

- In most cases the programmer will write at least one constructor for a class. This allows us to specify values for the initialised instance variables.
- These constructors are known as ***user defined constructors***.
- User defined constructors may initialise the object’s instance variables to values other than 0 or null.

User defined no-argument constructor

```
public class Pen

{
  // instance variable

  private double price;
  private String colour;

  // constructor

  public Pen()
  {
    price = 0.0;

    colour = "";
  }

  // methods

  …

}
```

```
public class PenTester
{

 public static void main(String[] args)
 {

   // create a Pen object

   Pen myPen = new Pen();
   …
   …

}
```

A user defined no-argument constructor

Constructors with input parameters

- Constructors can take input parameters.
- This allows the code which is calling the constructor to specify the values for initialisation of instance variables.
- For example, when creating a BankAccount object, we might want to specify an opening balance

```
BankAccount myAcc = new BankAccount(100);
```

Constructor example

File BankAccount.java

```
public class BankAccount
{

 // instance variables
 private double balance;

 // constructor
 public BankAccount(double openBal)
 {
    balance = openBal;
 }

}
```

File BankAccountTester.java

```
public class BankAccountTester
{

public static void main(String[] args)
{

// create a BankAccount object
BankAccount myAcc = new BankAccount(100);



}
}
```

Call to the Constructor

Problem…

File BankAccount.java

```
public class BankAccount
{

 // instance variables
 private double balance;

 // constructor
 public BankAccount(double openBal)
 {
    balance = openBal;
 }

}
```

File BankAccountTester.java

```
public class BankAccountTester
{

public static void main(String[] args)
{

// create a BankAccount object
BankAccount myAcc = new BankAccount();
```

Now this will cause an error – why?

```
}
}
```

Overloading constructors

- Like methods, constructors can be overloaded. This involves providing more than one constructor for a class.
- This means that objects can be instantiated in different ways, depending on the situation.
- For example, you could allow the creation of a new `BankAccount` object by specifying its opening balance, or you may want the balance to be 0.0
- The compiler will choose which constructor to execute depending on the number and type of input parameters. The different constructors ***must*** have different parameter lists.

Constructor overloading

File BankAccount.java

```
public class BankAccount
{
 // instance variables
 private double balance;

 // constructors
 public BankAccount()
 {
    balance = 0.0;
 }

 public BankAccount(double openBal)
 {
    balance = openBal;
 }…
}
```

File BankAccountTester.java

```
public class BankAccountTester
{

public static void main(String[] args)
{

// create a BankAccount object
BankAccount myAcc = new BankAccount();
BankAccount mySav =new BankAccount(1000);



}
}
```

Compiler knows which one to call based on number and type of parameters

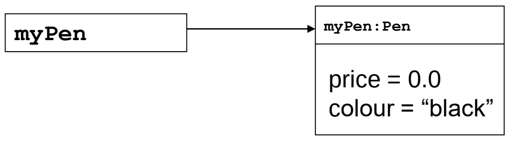
Constructor overloading

```
public class Pen

 {
   // instance variable

   private double price;
   private String colour;

   // constructor
   public Pen()
   {
     price = 0.0;
     colour = "black" ;
   }
   public Pen(String colourIn)
   {
     price = 0.0;
     colour = colourIn;
   }…
 }
```

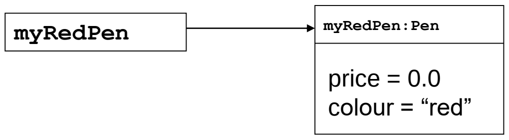
```
public class PenTester
{

 public static void main
                (String[] args)
 {

  // create a Pen object

  Pen myPen = new Pen();
  Pen myRedPen = new Pen("red");

  …
  …

}
```

Constructor overloading

```
Pen myPen = new Pen();
```



```
Pen myRedPen = new Pen(“red”);
```



Constructor overloading

```
public class Pen
{
  …
  // constructor
  public Pen()
  {
    price = 0.0;
    colour = “black” ;
  }
  public Pen(String colourIn)
  {
    price = 0.0;
    colour = colourIn;
  }
  public Pen(double priceIn,String colourIn)
  {
    price = priceIn;
    colour = colourIn;
  }…
}
```

```
public class PenTester
{

 public static void main
                    (String[] args)
 {

  // create a Pen object

  Pen myPen = new Pen();
  Pen myRedPen = new Pen(“red”);
  Pen p = new Pen(.55, “blue”);
  …
  …

 }

}
```

Compiler knows which one to call based on number and type of parameters

Constructor overloading

- All constructors of a class have the same name, the name of the class
- The compiler can tell them apart because they take different parameters – different signatures

```
public BankAccount()
public BankAccount(double initialAmount)
```

Consider..

All bank accounts should have an account number as well as a balance. However, it would be unadvisable to write a set method for accountNumber. Why?

How can you ensure that all BankAccounts will have an account number other than zero that cannot be changed throughout the lifetime of the BankAccount object?

Exercise

1. Write two constructors for a Book class which has instance variables title, author and year. The first constructor should take input parameters to set the title and author, and should set the year to 2021. The third constructor should take input parameters for all three instance variables.

2. Write a fragment of Java code that will show how each of these constructors would be invoked.