CSE1030 – Introduction to Computer Science II

Lecture #3

Non-Static Features
of Java Classes I

Goals for Today

- Goals:
  - Understand Objects Better
  - "Inherent Relationships"
- Practical: (Assignment #2!)
  - Features of a real Java class
  - Details, details, details… particularly:
    - Constructors
    - `main()`

CSE1030 – Lecture #3

- Review
- The Person Class – Holding Data
- The Default Constructor
- Grouping Data and Code Together
- Copy Constructors
- `Main()` as a Testing Facility
- We're Done!

Review: OOP Theory

- Big Ideas of Object Oriented Programming:
  - **Encapsulation**
    - Data & Code in single well-defined location
    - Hide complexity away, only expose a simple API
  - Takes Advantage of **Inherent Relationships**
    - In the Data, System, and Algorithms to be processed
- Java is an Object Oriented Language
  - In Java: **Everything** is an **Object**
  - A Java **Class** is how Objects are Defined

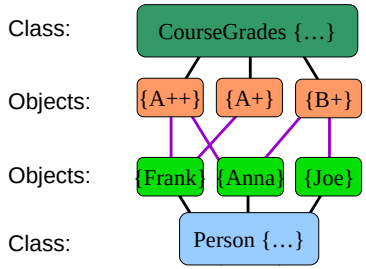
Review: Elements of a Java Class

1. Name the Class
2. How to Construct an Object
3. Who can use, and How
4. Defines the Data
5. Contains the Code

```
// any needed package statement
// any needed import statements

public class ClassName
{
    // data declarations
    private int i;

    // constructor
    ClassName(){ i = 0; };

    // method definitions
    int getI() { return i; }
    void setI(int pi) {i = pi; }
}
```


Review: How it Looks when Running

- Writing a Class is Building a Piece of the Mosaic that is your Program



CSE1030 – Lecture #3

- Review
- The Person Class – Holding Data
- The Default Constructor
- Grouping Data and Code Together
- Copy Constructors
- **`Main()`** as a Testing Facility
- We're Done!

Creating a Person Object

- Want to:
  - Employ OOP Philosophy of Good Organisation
  - Classes should reflect the **Inherent Relationships**
- Persons Have Attributes
  - Name
  - Age
  - Weight
  - etc.

Data / Attributes

- A lot of the OOP Philosophy has to do with Accessing and Changing the Data
- Advice: Keep Data **`private`**
- Allow Access via Accessor & Mutator Functions:
  - Accessors → getData() / Mutators → setData()
  - This gives the API creator Control
    - You can **Act** when something has Changed because you Made them **Call a Function**
  - Isolation & Implementation Independence
    - You can freely **Change** the **Implementation**
      No one will Know, No one will have to Change their Code!

The Person Class

```
public class Person
{
   // attributes
   private String Name;
   private int Age;

   // no constructors

   // methods
   public String getName()         {return Name;}
   public void   setName(String n) { Name = n; }

   public int  getAge()      {return Age;}
   public void setAge(int a) { Age = a; }
}
```


CSE1030 – Lecture #3

- Review
- The Person Class – Holding Data
- The Default Constructor
- Grouping Data and Code Together
- Copy Constructors
- **`Main()`** as a Testing Facility
- We're Done!

Constructors

- Person Class uses the Default Constructor
  - No Constructor ➔ Default Constructor
  - Default Constructor Initialises:
    - numerics = 0
    - booleans = false
    - objects = null
- Why would you use the Default Constructor?
  - Because it's Easy
  - Less Coding
- For simple Classes, this is Fine
  - But the Person Class is not Simple…

A Client who uses the Person Class

```
public class client
{
   public static void main(String[] args)
   {
      // First we want to create a person
      Person p = new Person();
      p.setName("William");
      p.setAge(36);

      // Do something with Person object...
      // ...
   }
}
```


Problem #1 – What is a Valid Person?

```
public class client
{
   public static void main(String[] args)
   {
      // First we want to create a person
      Person p = new Person();

      p.setName("William");

      p.setAge(36);

      // Do something with Person object...
      // ...
   }
}
```

What exists at this point?

What exists at this point?

Default Constructor Problems

- Incomplete Construction
  - Could Lead to Problems!
    Other Code may Assume objects have certain Properties
- Person Default Constructor forces the API user to know how to create an object
  - What if they forget to set the Age?
- **The Responsibility of Proper Construction rests with the API Code not with the User of the API Code!!**

A Better Person Class

```
public class Person
{
   private String Name;
   private int Age;

   // constructor
   Person(String name, int age)
      { Name = name; Age = age; }

   // methods
   // ...
}
```

Clients Must Use New Constructor

- The Old Code Doesn't Work Anymore!
  - javac client.java:

```
client.java:7: cannot find symbol
symbol  : constructor Person()
location: class Person
      Person p = new Person();
                 ^
1 error
```


Improved Client

- New constructor **Must** be used:

```
public class client
{
   public static void main(String[] args)
   {
      // First we want to create a person
      Person p = new Person("William", 36);

      // Do something with Person object...
      // ...
   }
}
```


CSE1030 – Lecture #3

- Review
- The Person Class – Holding Data
- The Default Constructor
- Grouping Data and Code Together
- Copy Constructors
- **`Main()`** as a Testing Facility
- We're Done!

Grouping Data & Code Together (1)

- Good Organisation supports even Large or Complex Programs
- Groups / Modules / Classes should reflect the **Inherent Relationships**
- Example: **Minimum Age to Drive**

One Possibility… Do it in the Client

```
public class client
{
   public static void main(String[] args)
   {
      // First we want to create a person
      Person p = new Person("William", 36);

      // Old enough to drive?
      if(p.getAge() >= 16)
         System.out.println(p.getName()
               + " May Apply for a "
               + "Driver's Licence");
   }
}
```

Output: `William May Apply for a Driver's Licence`

Grouping Data & Code Together (2)

- Not Well Organised
- Doesn't reflect the **Inherent Relationship**
  - Age testing **Directly Relates** to the **State** of a Person object, so it should be done **Inside the Object**
- Otherwise Users of API are Required to Know Internal Things, and to be Consistent
  - What if somebody forgets the driving age, or makes a mistake and uses the wrong age?

An Even Better Person Class

```
public class Person
{
   // attributes
   final int DrivingAge = 16;

   //...

   // methods
   //...
   public boolean mayDrive()
      { return Age >= DrivingAge; }
}
```


Much Better Client

```
public class client
{
   public static void main(String[] args)
   {
      // First we want to create a person
      Person p = new Person("William", 36);

      // Old enough to drive?
      if(p.mayDrive())
         System.out.println(p.getName()
               + " May Apply for a "
               + "Driver's Licence");
   }
}
```

Output: `William May Apply for a Driver's Licence`

Summarising Ideas

- The attribute `final` **defines a constant**
  - Something that will never change value
- `boolean mayDrive()`
  Encapsulates the Complexity
  - User of the API is Freed from Having to Know Things about Person Class
- Increases the Consistency

CSE1030 – Lecture #3

- Review
- The Person Class – Holding Data
- The Default Constructor
- Grouping Data and Code Together
- Copy Constructors
- `Main()` as a Testing Facility
- We're Done!

Client wants a Copy

```
public class client
{
   public static void main(String[] args)
   {
      // First we want to create a person
      Person p = new Person("William", 36);

      // I'm so great, there should be 2 of me...
      Person p2 = new Person(p.getName(),
                             p.getAge());
   }
}
```


Copying an Object - Problems?

- Sometimes you need to a Copy an Object
  - Maybe you're going to give one away to another piece of code you don't trust
  - Maybe you need to do something that will destroy or modify the object, but you still want the original
- But the Example Client way of copying:
  - requires the client to know how to build an object
  - **Assumes** that there is **No Hidden Private Data**
    Is that always a valid assumption? **(Weight?)**
  - Is clumsy (shouldn't it be easier than that)?

Person Class (Part 1)

```
public class Person
{
   // attributes
   private String Name;
   private int Age;
   private int Weight;
   final int DrivingAge = 16;

   // constructor (Assumes no Weight info available)
   Person(String name, int age)
      { Name = name; Age = age; Weight = -1; }

   // constructor (With Weight info)
   Person(String name, int age, int weight)
      { Name = name; Age = age; Weight = weight; }
```

Magic Number

Person Class (Part 2)

```
// copy constructor
  Person(Person p)
     { Name = p.Name; Age = p.Age; Weight = p.Weight; }

  // methods
  public String getName()         {return Name;}
  public void   setName(String n) { Name = n; }

  public int  getAge()      {return Age;}
  public void setAge(int a) { Age = a; }

  public void setWeight(int w) { Weight = w; }

  public boolean mayDrive() { return Age >= DrivingAge; }
}
```

Missing Accessor

Better Copying Client

```
public class client2
{
   public static void main(String[] args)
   {
      // First we want to create a person
      Person p = new Person("William", 36);

      // I'm so great, there should be 2 of me...
      // Note that, because of hidden attribute
      // weight, we can't copy the other way
      Person p2 = new Person(p);
   }
}
```


Overloaded Constructors

- More than 1 constructor!
  - Basic Constructor:
    `Person(String name, int age)`
  - More Advanced Constructor:
    `Person(String name, int age, int weight)`
  - Copy Constructor:
    `Person(Person p)`
- Overloading
  - Two functions with the same name?
    - They are different if their **Parameters are Different**
    - Terminology: **Method's Signature must be Unique**

CSE1030 – Lecture #3

- Review
- The Person Class – Holding Data
- The Default Constructor
- Grouping Data and Code Together
- Copy Constructors
- **`Main()`** as a Testing Facility
- We're Done!

`main()` as a Testing Facility

- Last Class we talked about Multiple **`main()`** functions (1 per class in program)
- We said:
  - Every class can have its own main
  - Only controlling class's main gets run at start
- Why?
  - Testing the individual classes!!
  - Include Testing Client **in** Person Class, not External

Testing Person Class in `main()`

```
public class Person
{
   // …stuff omitted for brevity…

   // Testing
   public static void main(String[] args)
   {
      Person p  = new Person("William", 36, 120);
      Person p2 = new Person(p);

      // Test Copy, Did it work?  Can check private Data!!
      if(p2.getName().equals("William")
      && p2.getAge() == 36
      && p2.Weight == 120)
         System.out.print("Copy Constructor Test Passed!");
      else
         System.err.print("Copy Constructor Test Failed!");
   }
}
```


`main()` for Testing – Summary

- **`main()`** is a part of the class, so
  - It has **Access to All Data and Code**
  - Even **Private** Data and Code
- Using main to do Unit Testing means
  - Your tests are **in one easy to find place**
  - And they are **With the Code** that they Test!

CSE1030 – Lecture #3

- Review
- The Person Class – Holding Data
- The Default Constructor
- Grouping Data and Code Together
- Copy Constructors
- **`Main()`** as a Testing Facility
- We're Done!

Next topic…

Non-Static Features
of Java Classes II