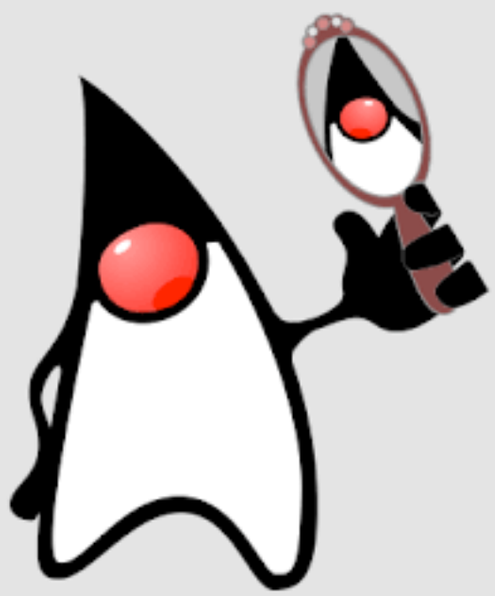
Objects and Classes

Abstraction and Encapsulation



COMP2603
Object Oriented Programming 1

Week 2, Lecture 1

Outline

- Abstraction
- Encapsulation
- Information Hiding
  - Access Modifiers
- Constructors
  - Instance Creation and Initialisation
- Message Passing Syntax
  - Method signature
  - Method overloading

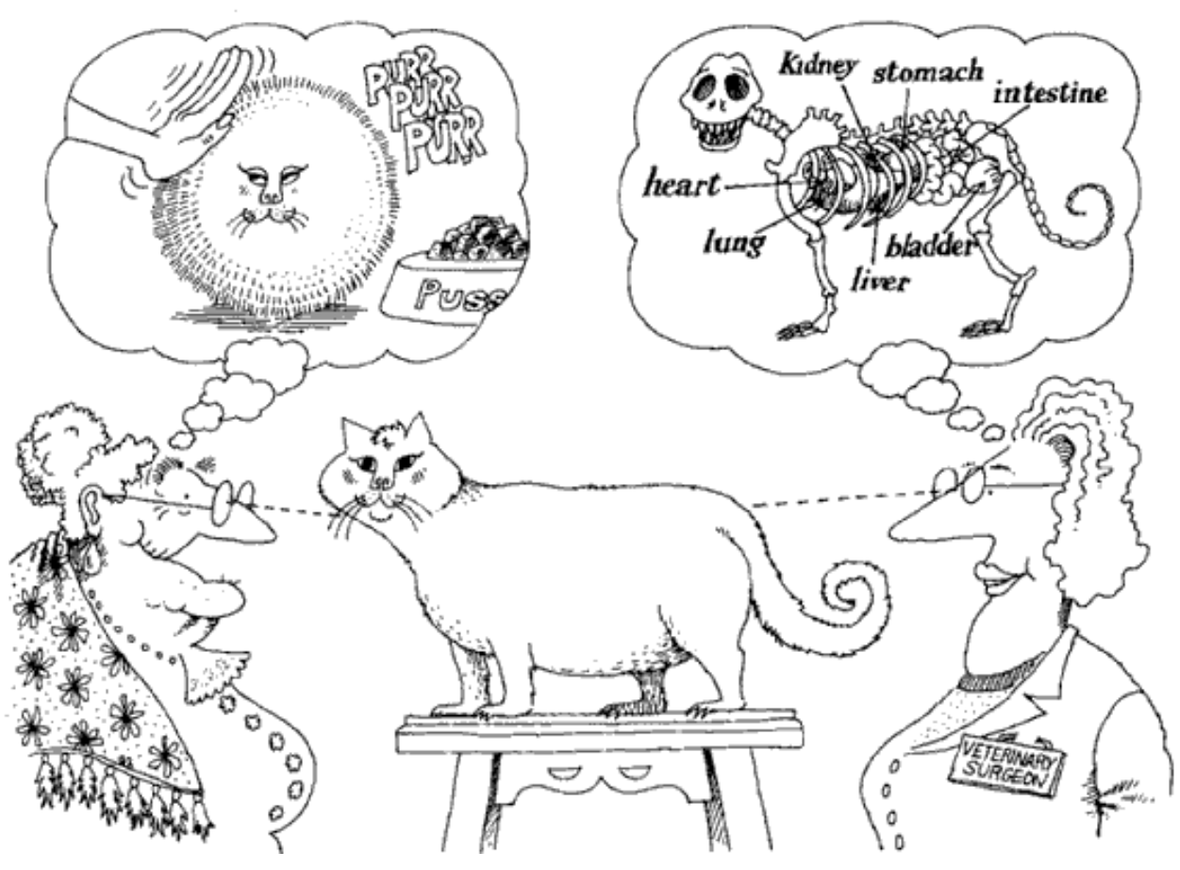
Abstraction

"Abstraction is one of the fundamental ways that we as humans cope with complexity." (Booch, 1994).

- Recognition of similarities
- Emphasis on significant details
- Independent of implementing mechanism

Abstraction

An abstraction denotes the **essential** characteristics of an object that **distinguish** it from all other kinds of objects and thus provide crisply defined **conceptual boundaries**, relative to the perspective of the viewer.



Observable behaviour

Example - Cat.java - state

```java
public class Cat{

    String name;
    double weight;
    int age;
    boolean hungry;
    String[] favouriteFoods;

}
```

Example - Cat.java - accessor

```java
public class Cat{

    String name;
    double weight;
    int age;
    boolean hungry;
    String[] favouriteFoods;

    //Accessors
    public String getName( ){ return name;}
    public double getWeight( ){ return weight;}
    public int getAge( ){ return age;}
    public boolean getHungry( ){ return hungry;}

}
```

Example - Cat.java - runner output

Cat × Runner ×

Compile Undo Cut Copy Paste Find... Close

```java
public class Runner{
    public static void main(String[] args){
        Cat cat = new Cat();
        String catName = cat.getName();
        System.out.println(catName);

    }
}
```

BlueJ: Terminal Window - Week2-2024

```
null
```

Example - Cat.java - mutators

```java
public class Cat{

    String name;
    double weight;
    int age;
    boolean hungry;
    String[] favouriteFoods;

    //Accessors
    public String getName( ){ return name;}
    public double getWeight( ){ return weight;}
    public int getAge( ){ return age;}
    public boolean getHungry( ){ return hungry;}

    //Mutators
    public void setName(String name){ this.name = name;}
    public void setWeight(double weight){ this.weight = weight;}
    public void setAge(int age){ this.age = age;}
    public void setHungry(boolean hungry){ this.hungry = hungry;}

}
```

Example - Cat.java - runner

Cat × Runner ×

Compile Undo Cut Copy Paste Find... Close

```java
public class Runner{
    public static void main(String[] args){
        Cat cat = new Cat();
        String catName = cat.getName();
        System.out.println(catName);

        cat.setName("Aarfield");
        catName = cat.getName();
        System.out.println(catName);
    }
}
```

BlueJ: Terminal Window - Week2-2024

```
null
Aarfield
```

Accessors and Mutators

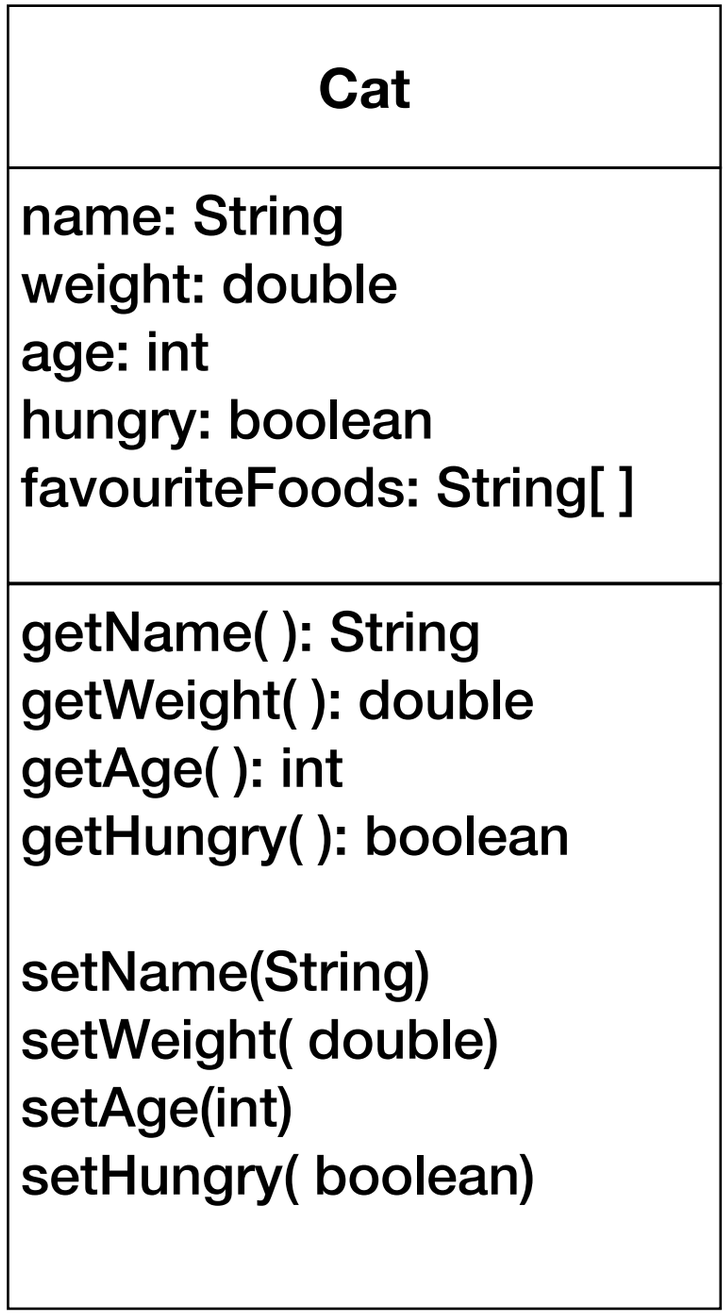
In object-oriented programming, class definitions usually provide `public` methods to *set* and *get* the values of its `private` instance variables.

Methods that **set** or **modify** an object's instance variables are called ***mutator methods*.**

Methods that **get** or **retrieve** the value of an instance variable are called ***accessor methods***.

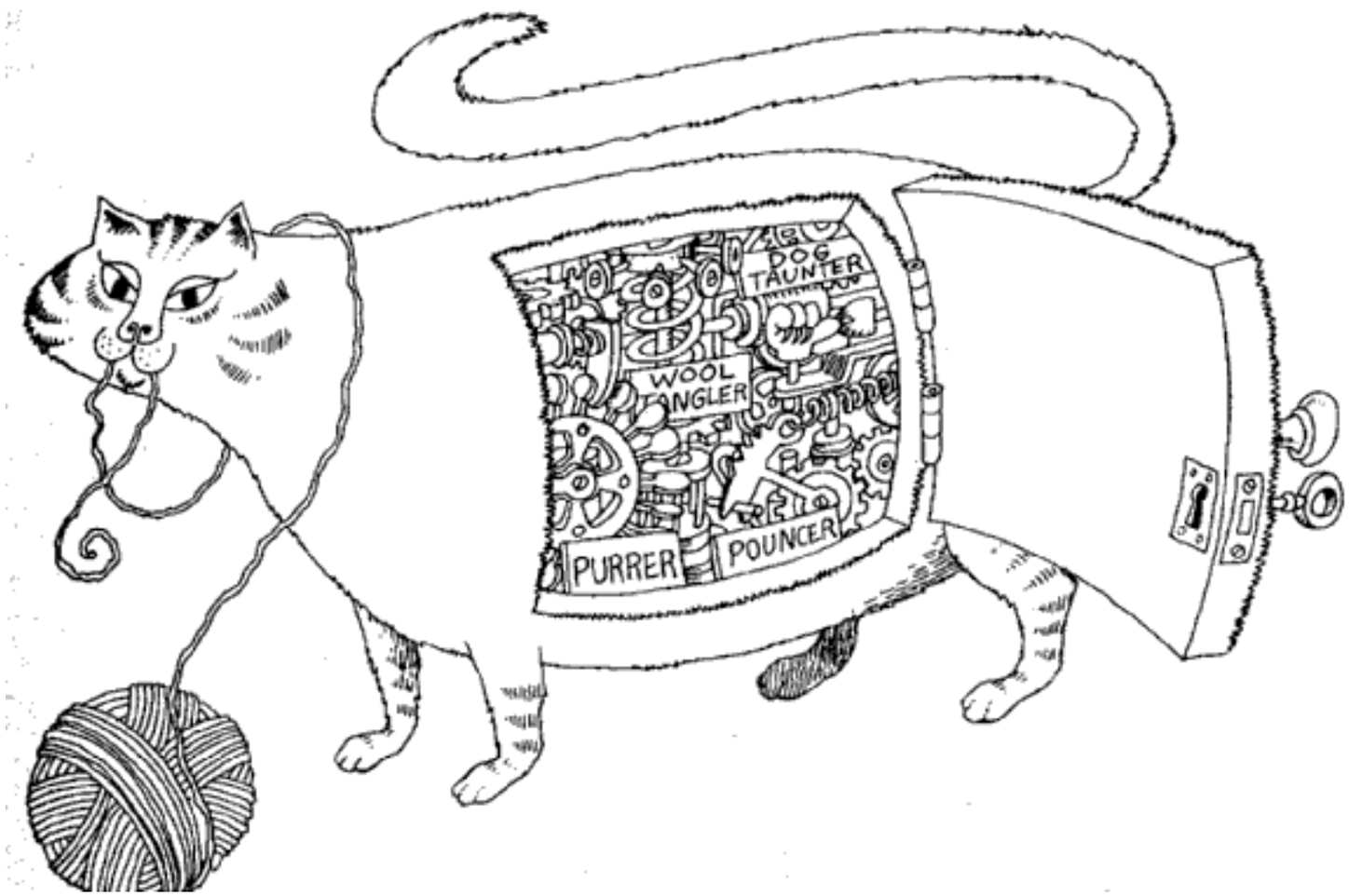
https://runestone.academy/ns/books/published/javajavajava/passing-info-to-object.html#accessors-mutators-3-2-1-1

UML - Accessors and Mutators



Encapsulation

Encapsulation focuses upon the implementation that gives rise to observable behaviours.



Hides the details (implementation) that gives rise to observable behaviour

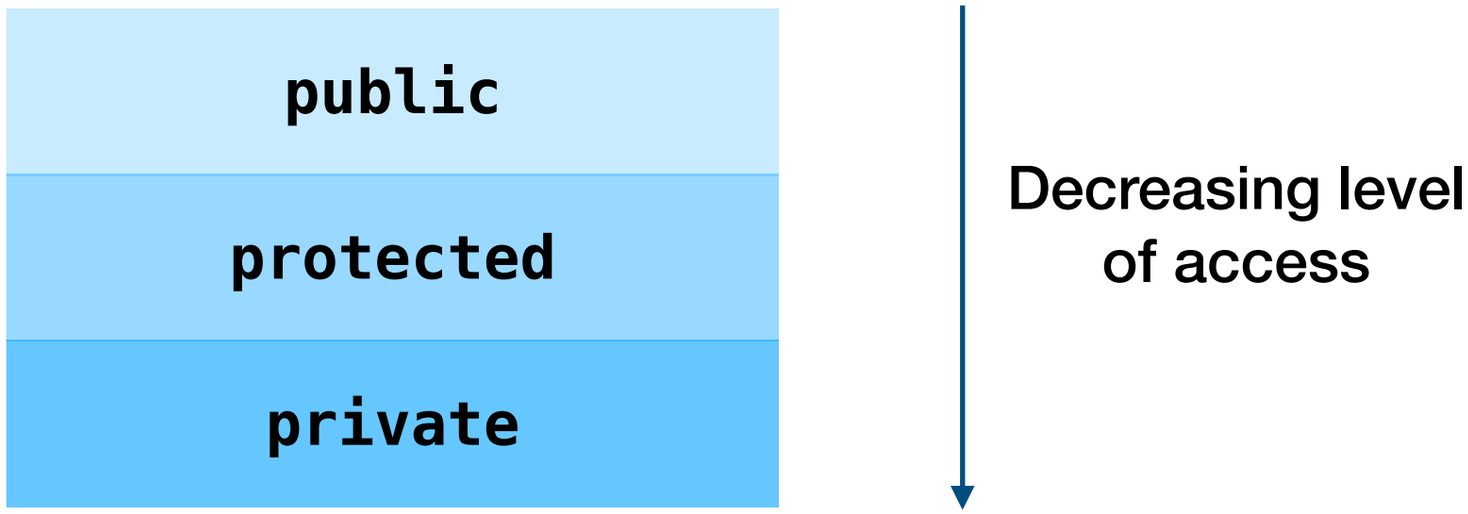
Information Hiding

Encapsulation is most often achieved through information hiding.

Information Hiding is the process of hiding all the secrets of an object that do not contribute to its essential characteristics:

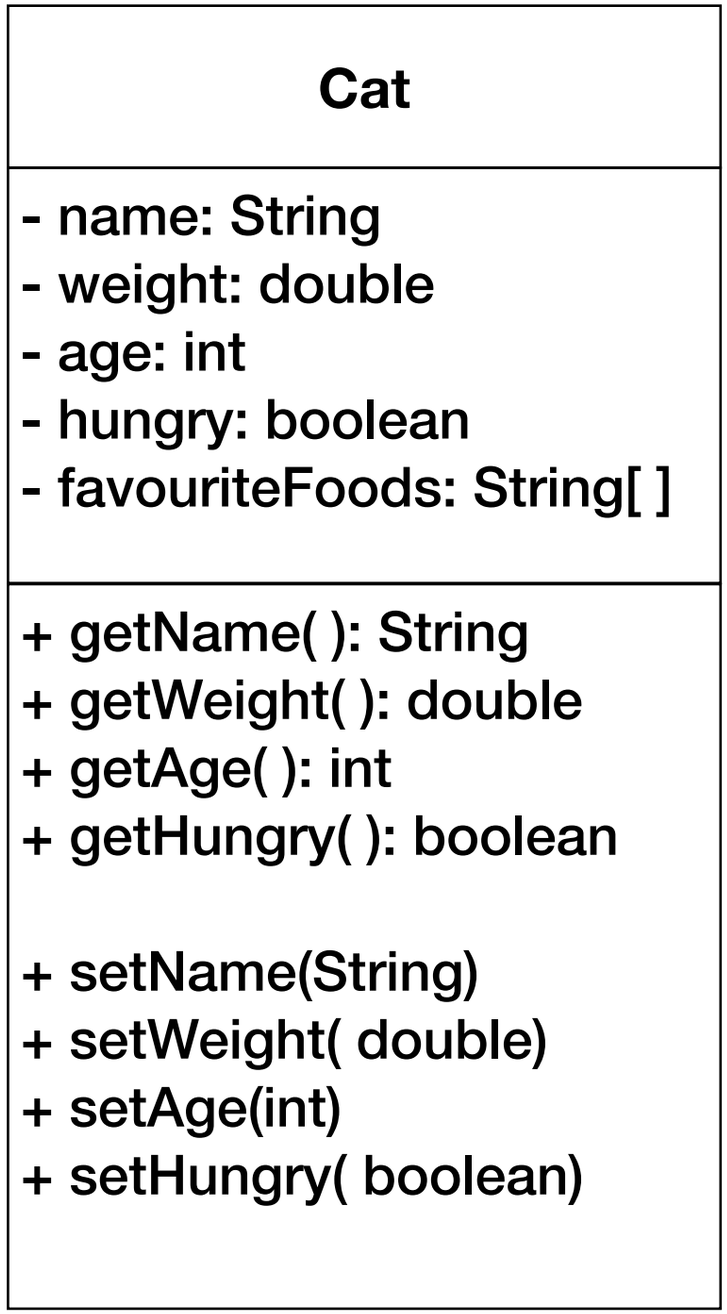
- the structure of an object is hidden
- the implementation of its methods is hidden

Access Modifiers



- Public: A declaration that is accessible to all clients
- Protected: A declaration that is accessible only to the class itself, its subclasses, and its friends
- Private: A declaration that is accessible only to the class itself and its friends

UML - Access Modifiers



`public`	+
`protected`	#
`private`	–

Example - Accessing State

Cat × Runner ×

Compile Undo Cut Copy Paste Find... Close Source Code

```java
public class Runner{
    public static void main(String[] args){
        Cat cat = new Cat();
        cat.setName("Aarfield");
        System.out.println(cat.getName() +" (via accessor)");
        System.out.println(cat.name      +" (raw reference)");
    }
}
```

BlueJ: Terminal Window - Week2-2024

```
Aarfield (via accessor)
Aarfield (raw reference)
```

Allowed because state has not been set to private

Example - Modifying State

Cat × Runner ×

Compile Undo Cut Copy Paste Find... Close Source Code

```java
public class Runner{
    public static void main(String[] args){
        Cat cat = new Cat();
        cat.setName("Aarfield");
        System.out.println(cat.getName() +" (via accessor)");
        System.out.println(cat.name      +" (raw reference)");
        cat.name = null;
        System.out.println(cat.getName() +" (via accessor)");
        System.out.println(cat.name      +" (raw reference)");
    }
}
```

BlueJ: Terminal Window - Week2-2024

```
Aarfield (via accessor)
Aarfield (raw reference)
null (via accessor)
null (raw reference)
```

Allowed because state has not been set to private. Bad design - no control over state

Example - Hiding State

Cat × Runner ×

Compile Undo Cut Copy Paste Find... Close Source Code

```java
public class Cat{

    private String name;
    private double weight;
    private int age;
    private boolean hungry;
    private String[] favouriteFoods;

    //Accessors
    public String getName( ){ return name;}
    public double getWeight( ){ return weight;}
    public int getAge( ){ return age;}
    public boolean getHungry( ){ return hungry;}

    //Mutators
    public void setName(String name){ this.name = name;}
    public void setWeight(double weight){ this.weight = weight;}
    public void setAge(int age){ this.age = age;}
    public void setHungry(boolean hungry){ this.hungry = hungry;}

}
```

Hidden because state has been set to private. Good design - full control over state

Example - Hiding State

Cat × Runner ×

Compile | Undo | Cut | Copy | Paste | Find... | Close | Source Code

```
1  public class Runner{
2      public static void main(String[] args){
3          Cat cat = new Cat();
4          cat.setName("Aarfield");
5          System.out.println(cat.getName() +" (via accessor)");
6          System.out.println(cat.name      +" (raw reference)");
7          cat.name = null;
8          System.out.println(cat.getName() +" (via accessor)");
9          System.out.println(cat.name      +" (raw reference)");
10     }
11 }
12
```

name has private access in Cat

Hidden because state has been set to private -> cannot be accessed freely

Example - Using Messages to Control State

Cat × Runner ×

Compile Undo Cut Copy Paste Find... Close Source Code

```java
public class Runner{
    public static void main(String[] args){
        Cat cat = new Cat();
        cat.setName("Aarfield");
        System.out.println(cat.getName() +" (via accessor)");
        //System.out.println(cat.name      +" (raw reference)"); <- NOT ALLOWED
        //cat.name = null; <- NOT ALLOWED
        System.out.println(cat.getName() +" (via accessor)");
        //System.out.println(cat.name      +" (raw reference)"); <- NOT ALLOWED
    }
}
```

Cat × Runner ×

Compile Undo Cut Copy Paste Find... Close Source Code

```java
public class Cat{

    private String name;
    //Accessors
    public String getName( ){ return name;}

    //Mutators
    public void setName(String name){
        if(name != null)
            this.name = name;
    }
```

Hidden because state has been set to private. Good design - full control over state

Constructors

Constructors are used to create an object and/or initialise its state.

- Constructors always carry the **same name** as the class.
- Constructors <u>do not</u> have a return type.
- Constructors generally should have parameters for attributes that client classes may be expected to send when the instance is created.

Default No-Argument Java Constructor

In Java, a default no-argument constructor is provided as part of the declaration of a class.

It does not take any arguments.

When you create a Java class and you do not specify a constructor explicitly, the default constructor will be used.

Its purpose is to allow creating an instance of a class.

https://runestone.academy/ns/books/published/javajavajava/constructors.html#sec-defaultconstructors-3

Example - Cat.java - Default Java Constructor

Cat × Runner ×

Compile Undo Cut Copy Paste Find... Close Source Code

```java
public class Cat{

    private String name;
    private double weight;
    private int age;
    private boolean hungry;
    private String[] favouriteFoods;

}
```

Cat × Runner ×

Compile Undo Cut Copy Paste Find... Close Source Code

```java
public class Runner{
    public static void main(String[] args){
        Cat cat = new Cat();
    }
}
```

User Defined Constructor - No Arguments

When you create a Java class and you specify a constructor explicitly, the default constructor will be **replaced** with the one that you supply.

Therefore, when creating an instance of a class, that newly defined constructor, if it has no arguments, will be used.

Example - Cat.java - No argument Constructor

Cat × Runner ×

Compile Undo Cut Copy Paste Find... Close Source Code

```java
public class Cat{

    private String name;
    private double weight;
    private int age;
    private boolean hungry;
    private String[] favouriteFoods;

    public Cat( ){
    }
}
```

The default no-argument constructor has been replaced even though the outcome is the same. Nothing special is being done in the constructor.

Cat × Runner ×

Compile Undo Cut Copy Paste Find... Close Source Code

```java
public class Runner{
    public static void main(String[] args){
        Cat cat = new Cat();
    }
}
```

User Defined Constructor with Parameters

When you create a Java class and you can specify a constructor that accepts parameters.

These parameters are used to initialise the state of an object when the constructor is invoked.

Usually, mutators are called in the constructor since they tend to have error checking code that is useful for validation of state values.

Example - Cat.java - Constructor with arguments

Cat × Runner ×

Compile Undo Cut Copy Paste Find... Close

```java
public class Cat{
    //State variables
    private String name;
    private double weight;
    private int age;
    private boolean hungry;
    private String[] favouriteFoods;

    // Constructor - One arg
    public Cat(String name){
        setName(name);
    }

    //Accessors
    public String getName( )     { return name;}
    public double getWeight( )   { return weight;}
    public int getAge( )         { return age;}
    public boolean getHungry( ) { return hungry;}

    //Mutators
    public void setName(String name){
        if(name != null)
            this.name = name;
    }
```

Example - Cat.java - Constructor with arguments

Cat × Runner ×

Compile Undo Cut Copy Paste Find... Close

```
1 public class Runner{
2     public static void main(String[] args){
3         Cat cat = new Cat();
4     }
5 }
6
7
8
```

constructor Cat in class Cat cannot be applied to given types;
required: java.lang.String
found: no arguments
reason: actual and formal argument lists differ in length

This constructor expects arguments now and throws a syntax error. The no-argument constructor has been replaced or overridden with one that expects a String

Cat × Runner ×

Compile Undo Cut Copy Paste Find... Close

```
1 public class Runner{
2     public static void main(String[] args){
3         Cat cat = new Cat("Aarfield");
4     }
5 }
6
7
8
```

Method Signature

Cat
- name: String - weight: double - age: int - hungry: boolean - favouriteFoods: String[ ]
+ getName( ): String + getWeight( ): double + getAge( ): int + getHungry( ): boolean + setName(String) + setWeight(double) + setAge(int) + setHungry(boolean)

A method causes certain actions to take place and can be regarded as a service provided by an object.

A method signature consists of the method's name, plus the number, types, and order of its formal parameters.

```
public String getName( ){
    return name;
}
```

https://runestone.academy/ns/books/published/javajavajava/constructors.html#sec-defaultconstructors-3

Method Signature

A class may not contain two methods with the same signature.”

Cat × Runner ×

Compile Undo Cut Copy Paste Find... Close

```
1  public class Cat{
2      //State variables
3      private String name;
4      private double weight;
5      private int age;
6      private boolean hungry;
7      private String[] favouriteFoods;
8
9      public String getName( )     { return name;}
10     public String getName( )     { return name.toUpperCase();}
11     public void getName( )       {   }
12
13
```

method getName() is already defined in class Cat

https://runestone.academy/ns/books/published/javajavajava/constructors.html#sec-defaultconstructors-3

Overloaded Methods

Two methods, however, can have the same name so long as the number, type, and order of its input parameters differ.

Cat × Runner ×

Compile Undo Cut Copy Paste Find... Close

```
//Mutators
public void setName(String name){
    if(name != null)
        this.name = name;
}
public void setName(String name, String title){
    if(name != null && title != null)
        this.name = title + " " + name;
}
public void setName(String name, String title, String suffix){
    if(name != null && title != null && suffix != null)
        this.name = title + " " + name + " " + suffix;
}
```

Example - Cat.java - Overloaded Methods

Cat × Runner ×

Compile Undo Cut Copy Paste Find... Close

```java
public class Runner{
    public static void main(String[] args){
        Cat cat = new Cat("Aarfield");
        System.out.println(cat.getName());
        cat.setName("Aarfield","Sir");
        System.out.println(cat.getName());
        cat.setName("Aarfield","Sir", "III");
        System.out.println(cat.getName());
    }
}
```

BlueJ: Terminal Window - Week2-2024

```
Aarfield
Sir Aarfield
Sir Aarfield III
```

Overloaded Constructors

Constructors are a special type of method (no return type, must share the class name etc) and so, they can be overloaded just like methods.

It is useful to have more than one constructor in a class since they allow flexibility when creating objects of the class.

When there are more than 1 constructor in a class, the second (and third and fourth etc) constructor is called an **overloaded constructor**.

Example - Cat.java - Overloaded Constructor

Cat × Runner ×

Compile Undo Cut Copy Paste Find... Close

```java
public class Cat{
    //State variables
    private String name;
    private double weight;
    private int age;
    private boolean hungry;
    private String[] favouriteFoods;

    //Constructor - Default No Argument
    public Cat( ){
    }
    // Constructor - One arg
    public Cat(String name){
        setName(name);
    }

```

Example - Cat.java - Overloaded Constructor

Cat × Runner ×

Compile Undo Cut Copy Paste Find... Close

```java
public class Cat{
    //State variables
    private String name;
    private double weight;
    private int age;
    private boolean hungry;
    private String[] favouriteFoods;
    //Constructor - Default No Argument
    public Cat( ){
    }
    //Overloaded Constructor - One arg
    public Cat(String name){
        setName(name);
    }
```

Cat × Runner ×

Compile Undo Cut Copy Paste Find... Close Source Code

```java
public class Runner{
    public static void main(String[] args){
        Cat cat = new Cat();
    }
}
```

Example - Cat.java - Overloaded Constructor

Cat × Runner ×

Compile | Undo | Cut | Copy | Paste | Find... | Close

```java
public class Runner{
    public static void main(String[] args){
        Cat cat = new Cat();
        System.out.println(cat.getName());
        Cat cat2 = new Cat("Heathcliff");
        System.out.println(cat2.getName());

    }
}
```

BlueJ: Terminal Window - Week2-2024

```
null
Heathcliff
```

Completed Code - Cat.java

Cat × Runner ×

Compile Undo Cut Copy Paste Find... Close

```java
public class Cat{
    //State variables
    private String name;
    private double weight;
    private int age;
    private boolean hungry;
    private String[] favouriteFoods;

    //Constructor - Default No Argument
    public Cat( ){
    }
        // Constructor - One arg
    public Cat(String name){
        setName(name);
    }

    //Accessors
    public String getName( )     { return name;}
    public double getWeight( )   { return weight;}
    public int getAge( )         { return age;}
    public boolean getHungry( ) { return hungry;}
    public void setWeight(double weight){
        if(weight > 0 && weight < 25) // lbs
            this.weight = weight;
    }
    public void setAge(int age){
        if(age > 0 && weight < 30)
            this.age = age;
    }
    public void setHungry(boolean hungry){
        this.hungry = hungry;
    }
        //Mutators
    public void setName(String name){
        if(name != null)
            this.name = name;
    }
    public void setName(String name, String title){
        if(name != null && title != null)
            this.name = title + " " + name;
    }
    public void setName(String name, String title, String suffix){
        if(name != null && title != null && suffix != null)
            this.name = title + " " + name + " " + suffix;
    }

}
```

Cat × Runner ×

Compile Undo Cut Copy Paste Find... Close

```java
public class Runner{
    public static void main(String[] args){
        Cat cat = new Cat();
        cat.setName("Aarfield");
        System.out.println(cat.getName() +" (via accessor)");
        //System.out.println(cat.name      +" (raw reference)"); <- NOT ALLOWED
        //cat.name = null; <- NOT ALLOWED
        System.out.println(cat.getName() +" (via accessor)");
        //System.out.println(cat.name      +" (raw reference)"); <- NOT ALLOWED

        System.out.println(cat.getName());
        cat.setName("Aarfield","Sir");
        System.out.println(cat.getName());
        cat.setName("Aarfield","Sir", "III");
        System.out.println(cat.getName());

        Cat cat1 = new Cat();
        System.out.println(cat1.getName());
        Cat cat2 = new Cat("Heathcliff");
        System.out.println(cat2.getName());

    }
}
```

toString( )

One of the most useful methods in a Java class is the `toString()` method.

The `toString()` method returns a String representation of its object.

Just as there is a default constructor, every Java class gets a default `toString()` which returns a string consisting of the name of the class of which the object is an instance, the at-sign character `@', and the unsigned hexadecimal representation of the hash code of the object.

https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Object.html#toString()

https://runestone.academy/ns/books/published/javajavajava/sec-tostring.html#sec-tostring-6

Default toString( ) - Cat.java

Cat × Runner ×

Compile Undo Cut Copy Paste Find... Close

```java
public class Runner{
    public static void main(String[] args){
        Cat cat = new Cat("Aarfield");
        String s = cat.toString();
        System.out.println(s);
    }
}
```

BlueJ: Terminal Window - Week2-2024

```
Cat@764dc6e3
```

Default toString( )- Cat.java - runner

Runner - Week2-2024

Cat × Runner ×

Compile Undo Cut Copy Paste Find... Close

```java
public class Runner{
    public static void main(String[] args){
        Cat cat1 = new Cat("Aarfield");
        Cat cat2 = new Cat("Heathcliff");

        String s1 = cat1.toString();
        String s2 = cat2.toString();

        System.out.println(s1);
        System.out.println(s2);

    }
}
```

BlueJ: Terminal Window - Week2-2024

```
Cat@5ae2e5c8
Cat@65bd0362
```

This method is meant to be replaced by one that produces a meaningful textual representation of the class.

Custom toString( ) - Cat.java

Cat - Week2-2024

Cat × Runner ×

Compile Undo Cut Copy Paste Find... Close Source Code

```java
public class Cat{
    //State variables
    private String name;
    private double weight;
    private int age;
    private boolean hungry;
    private String[] favouriteFoods;

    //Constructor - Default No Argument
    public Cat( ){
    }
        // Constructor - One arg
    public Cat(String name){
        setName(name);
    }

    //Accessors
    public String getName( )     { return name;}
    public double getWeight( )   { return weight;}
    public int getAge( )         { return age;}
    public boolean getHungry( ) { return hungry;}

    public String toString( ){
        return getName( )+ " , " + getAge() +  " , " + getWeight( ) +  " , " + getHungry( );
    }
```

Custom toString( ) - Cat.java - runner

Runner - Week2-2024

Cat × Runner ×

Compile Undo Cut Copy Paste Find... Close

```java
public class Runner{
    public static void main(String[] args){
        Cat cat1 = new Cat("Aarfield");
        Cat cat2 = new Cat("Heathcliff");

        String s1 = cat1.toString();
        String s2 = cat2.toString();

        System.out.println(s1);
        System.out.println(s2);

    }
}
```

BlueJ: Terminal Window - Week2-2024

```
Aarfield , 0 , 0.0 , false
Heathcliff , 0 , 0.0 , false
```

Cat.java - Random initialisation data

Cat - Week2-2024

Cat × Runner ×

Compile Undo Cut Copy Paste Find... Close Source Code

```java
public class Cat{
    //State variables
    private String name;
    private double weight;
    private int age;
    private boolean hungry;
    private String[] favouriteFoods;

    //Constructor - Default No Argument
    public Cat( ){

    }
        // Constructor - One arg
    public Cat(String name){
        setName(name);
        weight = Math.round(new java.util.Random( ).nextDouble(25));
        age = (new java.util.Random( )).nextInt(30);
    }

```

Custom toString( ) - Cat.java - runner

Runner - Week2-2024

Cat × Runner ×

Compile Undo Cut Copy Paste Find... Close

```java
public class Runner{
    public static void main(String[] args){
        Cat cat1 = new Cat("Aarfield");
        Cat cat2 = new Cat("Heathcliff");

        String s1 = cat1.toString();
        String s2 = cat2.toString();

        System.out.println(s1);
        System.out.println(s2);

    }
}
```

BlueJ: Terminal Window - Week2-2024

```
Aarfield , 6 , 11.0 , false
Heathcliff , 10 , 18.0 , false
```

Summary

Today you learned about:

- Abstraction
- Encapsulation/ Information Hiding
    - Accessors and Mutators
    - Access Modifiers
- Method Signatures
- Constructors
    - Overloaded vs Default
- toString( )

References

- Booch, G. (2007) Object-Oriented Analysis and Design. Chapter 2 - the Object Model
- Chapter 2 Objects: Using, Creating, and Defining: https://runestone.academy/ns/books/published/javajavajava/chapter-objects.html
- Chapter 3 Methods: Communicating With Objects: https://runestone.academy/ns/books/published/javajavajava/chapter-methods.html