Computer Programming

Introduction to Classes and Objects

Dr. Abdallah Mohamed

Computer Science

University of British Columbia

Previously…

- Java constructs and data types
  - **Variables**: `int`, `double`, `boolean`, …
  - **Displaying output**: `System.out.println()`
  - **Java Library**: `Scanner`, `Math`, `Character`
  - **Selection**: `if`, `if-else`, `switch`
  - **Loops**: `while`, `for`
  - **Methods**
- You used these concepts for simple, interesting programs
  - **Calculations**: area of a shape, unit conversion, …
  - **String processing:** reverse, isPalindrome, count letters, …
  - **Array processing**: sum, max, min, copying, …
  - **Simple games**: guess the number, paper-scissors-rock,…
  - **Project:** Jeopardy game
- **There are more interesting problems…**

Introduction to Objects

- The world consists of objects
  - Cars, people, places, animals, flowers, houses, chairs, etc

- We develop software to address issues in real-world
  - It makes sense to design software in terms of objects

What are ‘software’ objects?

- In a Java program, objects represent **entities in the real-world**
  - Each object has it’s **own space in the memory** to save information about this object.

What are objects?

- Let's look at one of these objects: the **farmer object**
- Any object:
  - has **attributes**: *define what the object is*
  - can perform **actions**: *define what the object can do*

*We build our software with **objects** that **work together** in order to **achieve the required goal***

Attributes

- …

Actions

- …

Object interaction

Attributes:

- **name**: Mark
- **weight**: 60.5 kg
- **location**: (20, 10)

Actions

- **Move right**
- **Move left**
- **Move up**
- **Move down**
- **Feed animal**

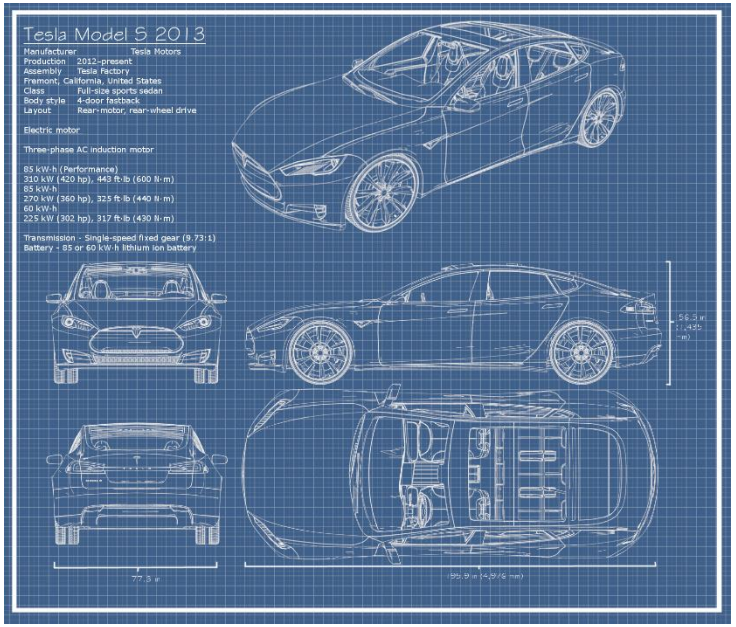
Today…

Aim: Learn how to

- **design objects,**
- **construct objects based on our design, and**
- **use objects**

Coding with objects

- **How are objects created in the real-world?**
  - TWO PHASES. Example: **Cars**.



Phase 1: Blueprint

- **Attributes**
- **Behaviour (Actions)**

Phase 2: Construction

In Java, all objects of a design have the same actions and attributes (although the attribute values can be different).

Learn by Example: The Farmer

Phase 1: Designing Objects

- A ***class*** represents the ***blueprint*** of a group of objects of the *same type*.

- This class defines the ***attributes*** and ***behaviors*** for objects.
  - **Attributes**
    - defined as variables inside our class
      - We call them "**instance variables**"

```
String name
double weight
int x, y
```

  - **Behavior (actions)**
    - defined as **methods** inside our class
      - Will discuss them later today!

Phase 1: Designing Objects, cont'd

- **Example**: the Farmer class

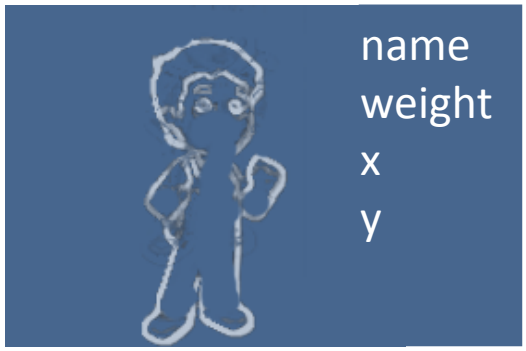
```
class Farmer {
      //instance variables (attributes)



      //methods (actions)

}
```

Famer Blueprint

Phase 1: Designing Objects, cont'd

- **Example**: the Farmer class

```
class Farmer {
      //instance variables (attributes)
      String name;
      double weight;
      int x, y;
      //methods (actions)
      //will add them later
}
```

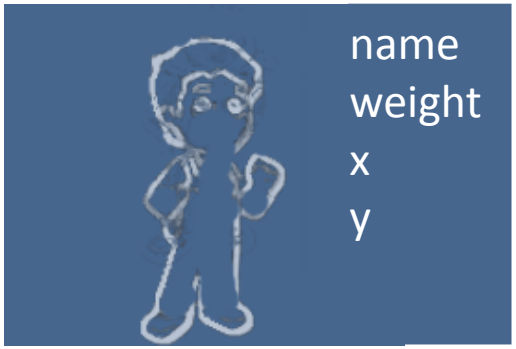


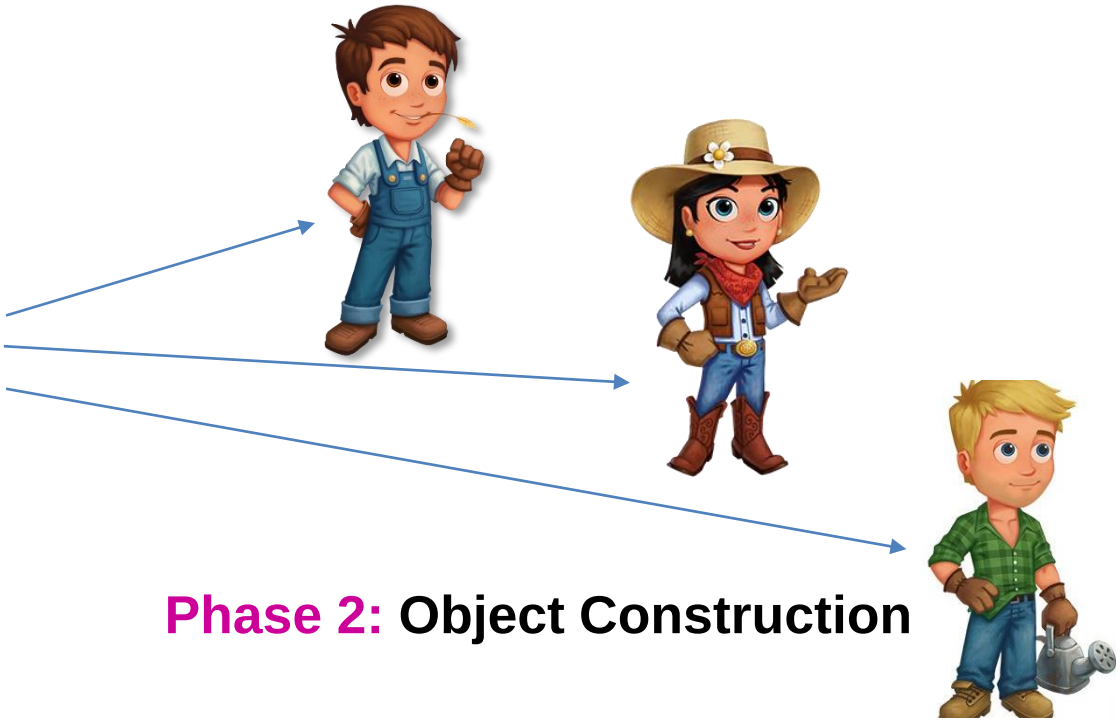
Famer Blueprint

- *Remember*: a class is just a blueprint for creating object
  - Classes store no data an perform no actions for an object
- We need to create objects now!

Phase 2: Creating and Using Objects

- Next, we need to create objects based on our class





Phase 1: Farmer Class

Phase 2: Object Construction

Phase 2: Creating objects using `new`

- Using the **`new`** keyword
- We will do this inside the **`main`** method

```
public class FarmerTest {
  public static void main(String[] args) {
      Farmer f1 = new Farmer();
  }
}
```

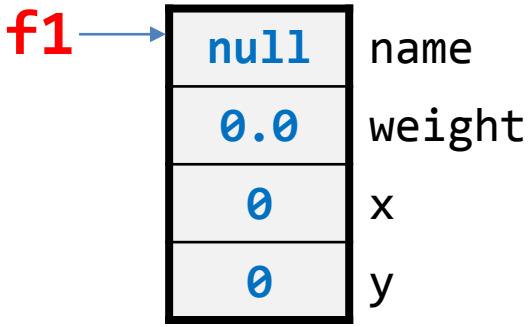
A unique identity for this object

Default values for the attributes

f1

- **name**: null
- **weight**: 0.0 kg
- **location**: (0, 0)

Each object has its own space in ***memory***



Phase 2: Creating and using objects

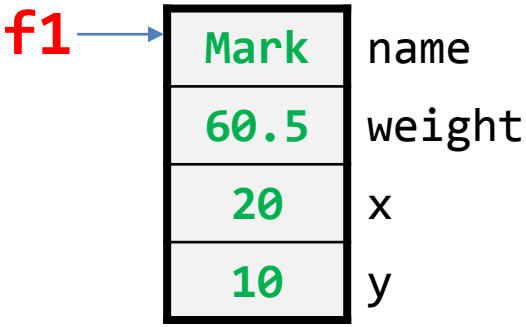
- Using the **new** keyword
- We will do this inside the **main** method

```
public class FarmerTest {
  public static void main(String[] args) {
      Farmer f1 = new Farmer();
      f1.name = "Mark";
      f1.weight = 60.5;
      f1.x = 20;
      f1.y = 10;
  }
}
```

After an object is created, its members can be accessed using the ***dot operator*** (.)

f1

- **name**: **Mark**
- **weight**: **60.5 kg**
- **location** : **(20 10)**



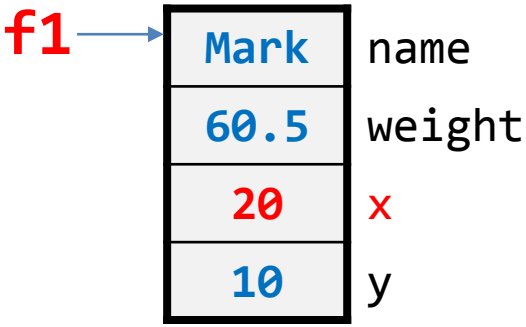
Phase 2: Creating and using objects

- Using the **new** keyword
- We will do this inside the **main** method

```
public class FarmerTest {
  public static void main(String[] args) {
      Farmer f1 = new Farmer();
      f1.name = "Mark";
      f1.weight = 60.5;
      f1.x = 20;
      f1.y = 10;
      System.out.printf( f1.x );
  }
}
```

f1

- **name**: Mark
- **weight**: 60.5 kg
- **location**: (20, 10)



Creating several objects

```
public class FarmerTest {
  public static void main(String[] args) {
      Farmer f1 = new Farmer();
      Farmer f2 = new Farmer();
      Farmer f3 = new Farmer();
      ... //change attribute values for f1,f2,f3
  }
}
```

f1 →	Mark	name
	60.5	weight
	20	x
	10	y

f2 →	Jessa	name
	51.4	weight
	17	x
	13	y

f3 →	John	name
	71.2	weight
	5	x
	19	y

Each object has its own memory space

Clicker Question

Consider the Farmer class. Which of the following is a valid instantiation of an object of the type Farmer?

```
class Farmer {
        String name;
        double weight;
        int x, y;
}
```

A. `Farmer f = Farmer();`

B. `Farmer = new Farmer();`

C. `Farmer f = new Farmer();`

D. `Farmer f = new Farmer("Mike");`

E. None of the above

Clicker Question

what is the value of the weight and name of the object **f** ?

```
class Farmer {
    String name;
    double weight;
}
```

```
public static void main(String[] args){
    Farmer f = new Farmer();
    name = "Mark";
    weight = 30;
}
```

A. "Mark", 30

B. null, 0

C. error

Practice

Assuming that we are developing a farm game where farmers need to feed their animals. An animal must be fed otherwise it becomes dead.

Create a class **`Cow`**:

- A cow has the attributes
  - **nickname** (`String`)
  - **stomach** (`int`) that percentage (0 to 100) of food in cow's stomach
  - **isFull** (`boolean`) that indicates whether the cow is full

Write a program to

- Create two Cow instances (objects) set their attributes to any values
- Display the information of the two Cow instances.

Adding Behaviour to Our Design

Updating Our Design

- The ***blueprint*** of a group of objects of the *same type* is represented by a ***class***.

- The class defines the ***attributes*** and ***behaviors*** for objects.
  - **Attributes** ✓
    - defined as variables inside our class
      - We call them "instance variables"
  - **Behavior (actions)**
    - defined as **methods** inside our class

Attributes

```
String name
double weight
int x, y
```

Actions

Move right
Move left
Move up
Move down

Adding Behaviour to Our Design

- **Example**: the Farmer class

```
class Farmer {
      //instance variables (attributes)
      String name;
      double weight;
      int x, y;

      //methods (actions)
      public void moveUp()    {y++;}
      public void moveDown()  {y--;}
      public void moveRight(){x++;}
      public void moveLeft()  {x--;}
}
```

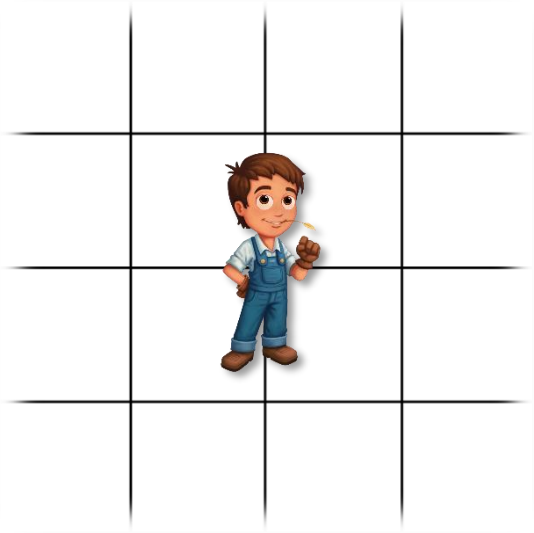
Note: methods inside a class can reference instance variables without (.)

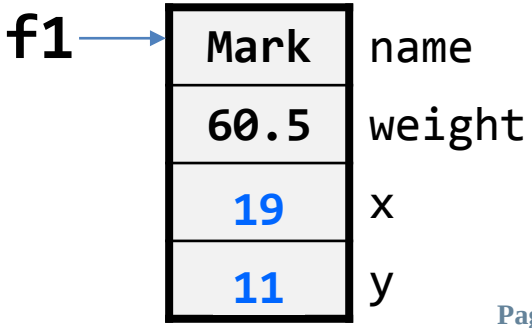
Using the updated design

- Using the **new** keyword
- We will do this inside the **main** method

```
public class FarmerTest {
  public static void main(String[] args) {
      Farmer f1 = new Farmer();
      f1.name = "Mark";
      f1.weight = 60.5;
      f1.x = 20;
      f1.y = 10;
      f1.moveRight();
      f1.moveDown();
      f1.moveTo(19,11);
  }
}
```

Once implemented, the farmer will learn this new action (see next slide)





Adding Behaviour to Our Design

- **Example**: the Farmer class

```
class Farmer {
      //instance variables (attributes)
      String name;
      double weight;
      int x, y;

      //methods (actions)
      public void moveUp()    {y++;}
      public void moveDown()  {y--;}
      public void moveRight(){x++;}
      public void moveLeft()  {x--;}
      public void moveTo(int a, int b){
         x = a;     y = b;
      }
}
```

Practice

(1) Modify your **`Cow`** class to include the following two methods:

- `void eat(int amount)` that increments food in `stomach` by the given amount.
- `void say(String msg)` that causes the animal to display the given `msg` on the console preceded by its `nickname`.
  For example, if the nickname is "`Bolt`" and `msg` is "`Hi`", the output is
  - **`Bolt says: Hi!`**

Attributes

- String nickname
- int stomach
- boolean isFull

Methods

- eat(…)
- say(…)

(2) Modify `eat` method such that `stomach` is never larger than a 100 at which `isFull` is set to true. Also, make sure the cow can't eat anymore if it is full (i.e., `isFull = true`).

Constructors

Constructors

- **What if I want to initialize objects as I create them?**
  - **Example:**

```
Farmer f1 = new Farmer();

f1.name = "Mark";
f1.weight = 60.5;
f1.x = 20;
f1.y = 10;
```

```
Farmer f1 = new Farmer("Mark", 60.5, 20, 10);
```

Constructors, cont'd

- Constructors play the role of **initializing objects**.

- Constructors are a **special kind of method.**

- They have 3 peculiarities:
  - Constructors must have the **same name as the class itself**.
  - Constructors **do not have a return type** -- not even void.
  - Constructors are invoked using the **`new`** operator **when an object is created**.

Constructors: Example

- **Example**: the Farmer class

```
class Farmer {
      //instance variables
      String name;
      double weight;
      int x, y;
      //constructors
      Farmer(String aName, int aWeight, int x1, int y1){
         name = aName;
         weight = aWeight;
         x = x1;
         y = y1;
      }
      //methods
      public void moveUp()    {y++;}
      public void moveDown()  {y--;}
      public void moveRight(){x++;}
      public void moveLeft()  {x--;}
      public void moveTo(int a, int b) { x = a; y = b; }
}
```

Constructors: Example

```
public class FarmerTest {
  public static void main(String[] args) {
      Farmer f1 = new Farmer("Mark", 60.5, 20, 10);
      Farmer f2 = new Farmer("Jessa", 51.4, 17, 13);
      Farmer f3 = new Farmer("John", 71.2, 5, 19);
  }
}
```

f1 →	Mark	name
	60.5	weight
	20	x
	10	y

f2 →	Jessa	name
	51.4	weight
	17	x
	13	y

f3 →	John	name
	71.2	weight
	5	x
	19	y

Try this now…

```
class Farmer {
      //instance variables
      ...
      //constructors
      Farmer(String aName, int aWeight, int x1, int y1){
         name = aName;
         weight = aWeight;
         x = x1;
         y = y1;
      }
      //methods
      ...
      ...
      public static void main(String[] args) {
            Farmer f1 = new Farmer("Mark", 60.5, 20, 10);
            Farmer f2 = new Farmer("Jessa", 51.4, 17, 13);
            Farmer f3 = new Farmer();  // ERROR!! WHY??
      }
}
```

The Default Constructor

- A **default constructor** is provided automatically only if no constructors are explicitly defined in the class.
- It sets the attributes to their default values:
  - String → null
  - Numeric → zero
  - Boolean → false

- In the previous example, the programmer included a four-argument constructor, and hence the default constructor was not provided.

Problem Fixed!!

```
class Farmer {
      //instance variables
      ...
      //constructors
      Farmer(){    //now we have a zero-arg constructor
      }
      Farmer(String fname, int fweight, int fx, int fy){
         name = fname;
         weight = fweight;
         x = fx;
         y = fy;
      }
      //methods
      ...
      public static void main(String[] args) {
            Farmer f1 = new Farmer("Mark", 60.5, 20, 10);
            Farmer f2 = new Farmer("Jessa", 51.4, 17, 13);
            Farmer f3 = new Farmer();  // No error!!
      }
}
```

Practice

Add two constructors to your **`Cow`** class:

- A zero-argument constructor to set the `stomach` to 50 and `nickname` to "Anonymous".
- A two-argument constructor to set the cow's `nickname` and `stomach` to given values. Make sure `stomach` doesn't get a value larger than 100.

Note that `isFull` is always set based on the value of `stomach`.

Test your class by creating a Cow instance with (`stomach = 30, nickname=Bolt`), make it eat 10 food units, and then make it say something like "Hi".

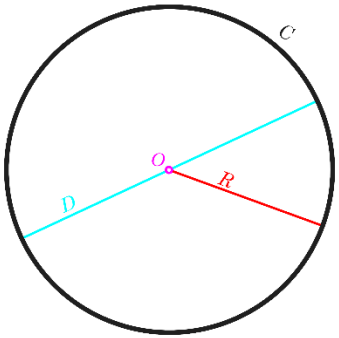
Attributes

- String nickname
- int stomach
- boolean isFull

Methods

- Cow()
- Cow(…)
- eat(…)
- say(…)

Try this at home!



- Write a class **`Circle`** which has:
  - an instance variable (attribute) `double radius`.
  - a no-argument constructor that sets `radius` to 10.
  - a one-argument constructor that sets `radius` to a given value.
  - a method `setRadius` that changes the radius to a given value.
  - two methods `getArea` and `getPerimeter` that return the area and perimeter respectively.

- Test your class by creating three instances of `Circle` and invoke their different methods.

UML Notation

- UML stands for Unified Modeling Language
- UML diagrams are one method for representing and communicating a *model* of the software being developed.

Farmer ← Class name

```
name: String
weight: double
x: int
y: int
```

← Attributes (data fields)

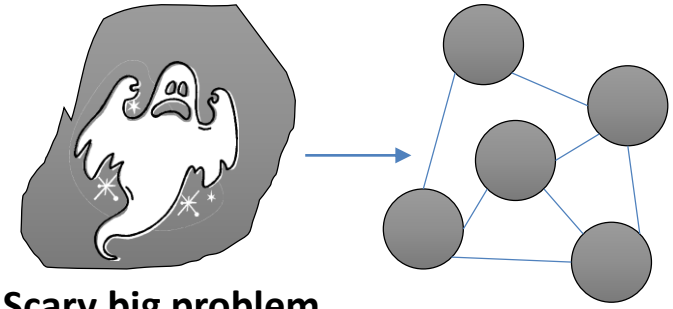
```
Farmer()
Farmer(name: String, weight: double, x:int, y:int)
moveUp(): void
moveDown(): void
moveRight(): void
moveLeft(): void
moveTo(int x, int y): void
```

← Methods

Advantages of Object Oriented Programming (OOP)

Modularization

- Big problem into smaller subproblems.
- Improves understandability



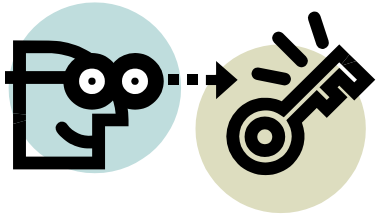
Scary big problem

Encapsulation and Reuse

- Hide complexity and protect low-level functionality.
- Reuse code in other programs

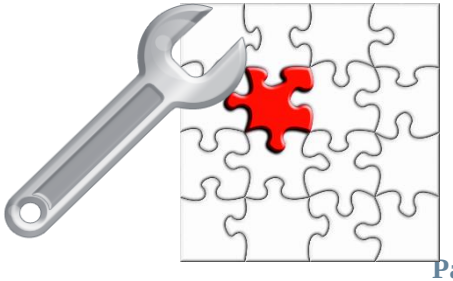
Understandability (abstraction)

- Composability (big objects are built off smaller ones)
  - More about this later



Maintenance

- Easier to change code of individual modules



Today's Objectives

- Definitions:
  - **Object-oriented programming (OOP):** a programming paradigm based on the concept of "objects"
  - **Class**: an object blueprint with methods and attributes
  - **Object**: an instance of a class that represents an entity of in th real world. An object has
    - a unique **identity**,
    - **state** defined in term of instance variables (aka properties or attributes)
    - **behavior (methods):** what the **object can do**.
  - **Instance variable (property)**: an attribute of objects
  - **Methods**:a set of statements that performs an action.
- Defining a class with instance variables and methods
- Purpose, use, and definition of constructors.
- Creating objects using `new`
- Calling object's methods using the dot (.) operator

Next time…

- More details about classes and objects
  - Data encapsulation
    - Accessor and mutator methods
  - Visibility modifiers
    - public, private, …
  - More about reference variables
    - reference variables vs. objects
    - reference variables vs. primitive ones.
  - Inheritance