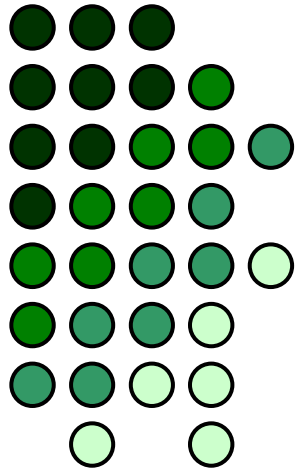
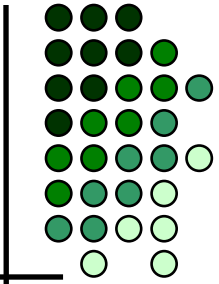
Java Objects

Paul Inventado

De La Salle University

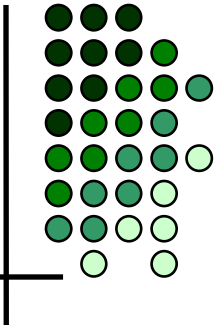


Java Objects



- Objects are organized by placing them inside packages (or directories).
- For objects to be created, classes should be in the same directory or package.
- If objects are present in other directories or packages they should first be *imported* using the *import* keyword.

Java Objects



- To include a class in a package, use the *package* keyword.
- A package may contain other packages and can be indicated using a dot “.” notation.

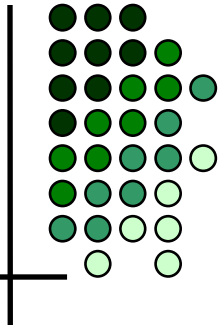
package <location>

package myPackage;

package myPackage.Level1;

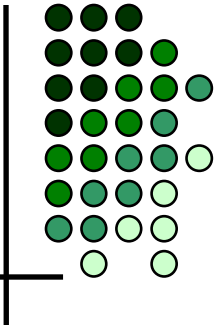
Package myPackage.Level1.Level2;

Java Objects



```
package myPackage;
public class Person
{
   private String sName;
   private int nAge;
   private String getName();
   {
        …
   }
}
```

Java Objects



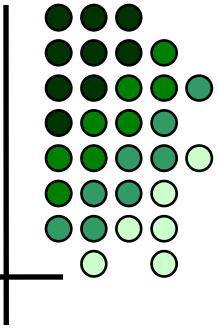
- To import from other packages, use the import keyword.
- Packages within other packages are accessed also using the dot “.” notation.
- To access all classes in a package use the “*” symbol.

import <Identifier and location of class>

import myPackage.Person;

import myPackage.*;

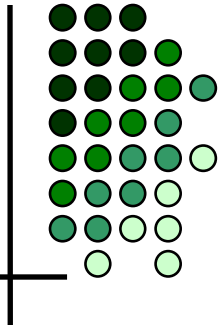
Java Objects



```
package myPackage;
public class Person
{
    private String sName;
    private int nAge;
    private String getName();
    {
      …
    }
}
```

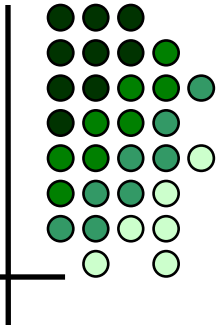
```
package myOtherPackage
import myPackage.Person;
public class myClass
{
    public static void
    main(String[] args)
}
```

Java Objects



Access Modifier	Class Itself	Subclass	Package	World
Public	✓	✓	✓	✓
Protected	✓	✓	✓	
Private	✓			

Java Objects



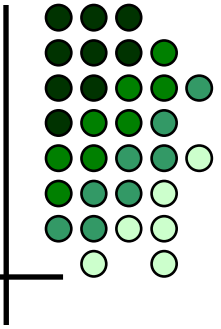
- To instantiate the objects use the *new* keyword.

<ClassName> Identifier = new <ClassConstructor>

Class myClass=new Class();

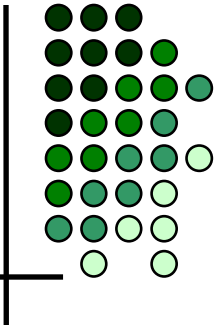
```
package myPackage;
public Driver
{
    public static void main(String[] args)
    {
     Person pGeorge=new Person();
    }
}
```

Java Objects



- An object contains all the attributes and operations defined in its class.
- The values of its attributes can be modified and its operations can be used.
- The “.” notation is used for accessing a class’ attributes and methods.

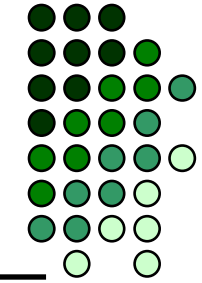
Java Objects



How is this Class represented in Java?

Person
-name: String -age: int
+ getName(): String + getAge(): int + setName(String): void + setAge(int): void

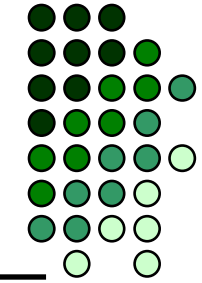
Java Objects



```
public myClass
{
    public static void main(String[] args)
    {
     Person pGeorge=new Person();
     pGeorge.setName("Dr. Seuss");
     pGeorge.setAge(23);
     System.out.println("Name:" + pGeorge.getName());
     System.out.println("Age:" + pGeorge.getAge());
    }
}
```

```
Name: Dr. Seuss
Age: 23
```

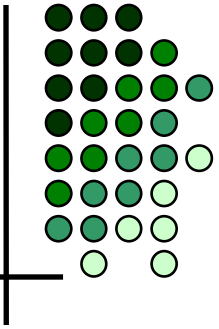
Java Objects



Name: Dr. Seuss
Age: 23

```
public Driver
{
    public static void main(String[] args)
    {
     Person pGeorge=new Person();
     pGeorge.name="Dr. Seuss";
     pGeorge.age=23;
     System.out.println("Name:" + pGeorge.name);
     System.out.println("Age:" + pGeorge.age);
    }
}
```

Java Objects

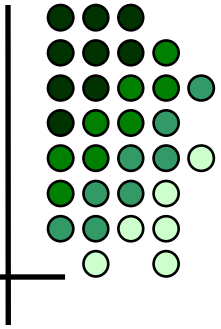


- **Each object exists on its own and has attributes and operations of its own.**

```
public myClass
{
    public static void main(String[] args)
    {
     Person pGeorge=new Person();
     Person pRinggo=new Person();
     pGeorge.setAge(23);
     System.out.println("Age: "+ pGeorge.getAge());
     pRinggo.nAge=26;
     System.out.println("Age: "+ pRinggo.getAge());
    }
}
```

```
Age: 23
Age: 26
```

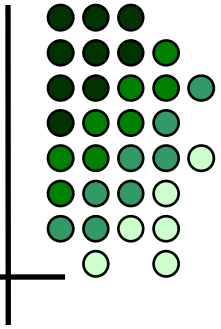
Java Objects



Constructor

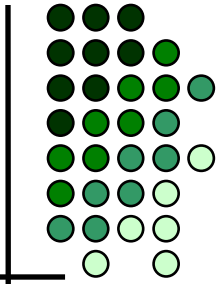
- Special type of method that is automatically executed when an object is created from the class
- Used to “construct” the initial state of an object
- Special method that initializes the instance variables

Java Objects



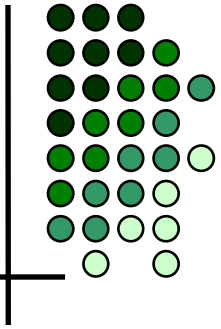
```
[accessmodifier] class <classname>
{
    [accessmodifier] <classname>([parameters])
    {
      //body
    }
}

public class myClass
{
    public myClass()
    {
      …
    }
}
```

Java Objects



```
public class Point
{    private int x;
     private int y;

     public Point()
     {  x = 0;   y = 0;   }
     public void setPoint(int a, int b)
     {x = a;   y = b;     }
     public int getX()  {  return (x);}
     public int getY()  {  return (y);}
     public void toString()
     {  System.out.println("x = " + x + "; y = " + y);
     }
}
```

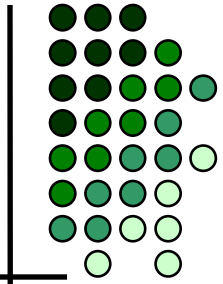
Java Objects



Rules for constructors

- Constructor name must have the same name as the class
- Constructor does not specify a return data type
- When a program instantiates an object of the class, the program can supply initializers in parenthesis to the right of the class name that are passed as arguments to the class' constructor

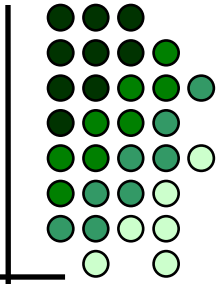
Java Objects



Constructor Overloading

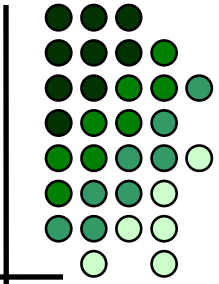
- Allows different ways to instantiate objects
- The constructor having the same parameter prototype as the actual parameter will be used
- The only differences between overloads of a constructor are its parameters
- Constructors having exactly the same parameters are not allowed

Java Objects



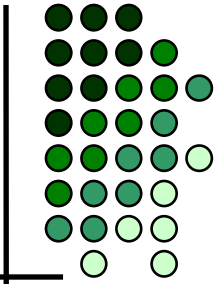
```
public class Person
{    private String name;
     private int age;

     public Person (String n)
     {  setname(n);  }
     public Person (int myAge)
     {   age = myAge;  }
     public Person (String n, int a)
     {  name=n; age = a; }
     public void setName (String n1)
     { name = n1;   }
}
```

Java Objects



- Attempting to declare a return data type for a constructor and/or attempting to return a value from a constructor is a logical error.
- Java allows other methods of the class to have the same name as the class and to specify return types. However, such methods are not constructors and will not be called when an object of the class is instantiated.

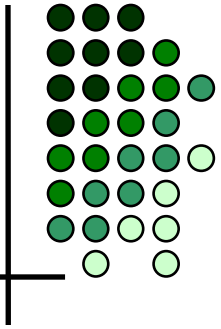
Java Objects



```
public class Person
{  private String name;
   private int age;

   public void Person (String n)
   {  name = n;  }
}
```

Not a Constructor

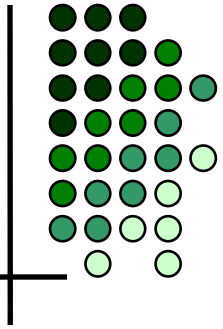
Java Objects



Default Constructor

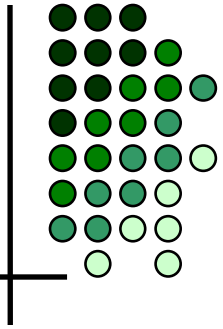
- Also referred to as no-argument constructor
- Automatically provided by Java IF no constructors are defined explicitly for the class
- Not inherited from the superclass
- Cannot be inherited by subclasses

Java Objects



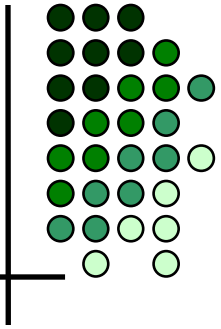
- Create a class Rectangle. The class has attributes length and width, each of which defaults to 1.0. It has methods that calculate the perimeter and area of the rectangle. It has get and set methods for both length and width. The set methods should verify that the length and width are floating-point numbers larger than 0.0 and less than 20.0. If not, it is set to 1.0

Java Objects



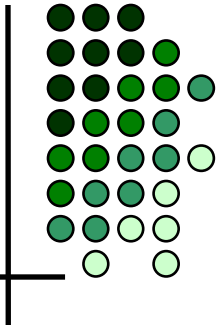
```
public class Rectangle {
    private float fLength;
    private float fWidth;

    public Rectangle()
    {
     fLength=1.0f;
     fWidth=1.0f;
    }

    public Rectangle(float length)
    {
     fLength=length;
     fWidth=1.0f;
    }
```

Java Objects



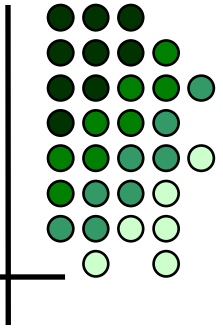
```
public Rectangle(float length, float width)
    {
     fLength=length;
     fWidth=width;
    }
    public double getLength() {
     return fLength;
    }
    public void setLength(float length) {
     if(length>=1 && length <=20)
          fLength = length;
     else
          fLength=1;
    }
```

Java Objects



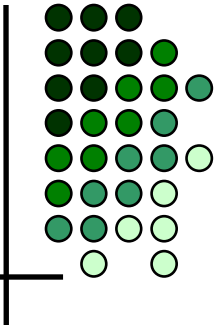
```
public double getWidth() {
     return fWidth;
    }
    public void setWidth(float width) {
     if(width >=1 && width <=20)
          fWidth = width;
     else
          fWidth=1;
    }
```

Java Objects



```
public double getPerimeter()
    {
     return fLength*2+fWidth*2;
    }

    public double getArea()
    {
     return fLength*fWidth;
    }
}
```

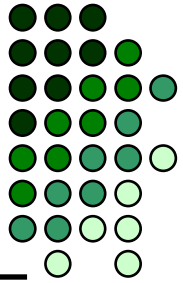
Java Objects



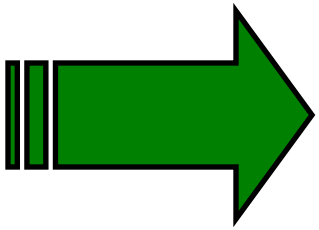
this keyword

- Calling other constructors
  - To remove redundancy, constructors may call other constructors
  - This is achieved by using the **this()** method

Java Objects

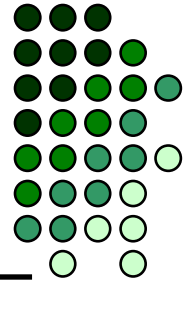


```
public class Point
{
    private int x;
    priate int y;
    public Test(int myX)
    {
        x=myX;
        y=0;
    }
    public Test(int myX, int myY)
    {
        x=myX
        y=myY;
    }
}
```



```
public class Point
{
    private int x;
    priate int y;
    public Test(int myX)
    {
        this(myX,0);
    }
    public Test(int myX, int myY)
    {
        x=myX;
        y=myY;
    }
}
```

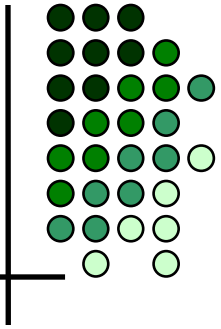
Java Objects



this keyword

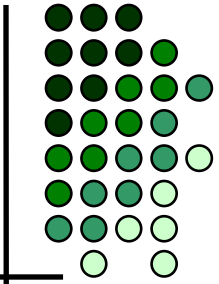
- Used to refer to the class where it is referenced
- Usually used to refer to class' attributes or methods
- Used to remove ambiguity with attribute and parameter names

Java Objects



```
public class Point
{
    private int x;
    private int y;

    public int getX()
    {
        return this.x;              //same effect as → return x;
    }
    public void setX(int x)
    {
            this.x=x;
    }
}
```

Java Objects

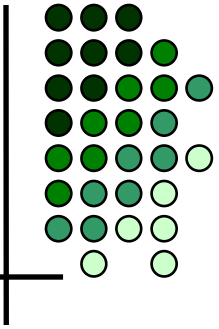


A *destructor* is executed once an object is destroyed. An object is destroyed when no other object is referencing it, or when it falls out of scope. The destructor is usually used to perform some cleanup tasks such as closing open files.

As opposed to constructors,

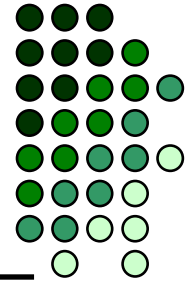
- A destructor's name is specified through the **`finalize`** keyword;
- A destructor always has a return type of **`void`**; and
- There can only be one destructor in a class.

Java Objects



```
public void finalize()
{
  count--;
  // close file
  // release object
}
```

Java Objects



- Java performs automatic garbage collection to help return memory back to the system
- The programmer may “suggest” to Java to perform garbage collection through the statement:

```
System.gc();
```

Java Objects

Paul Inventado

De La Salle University