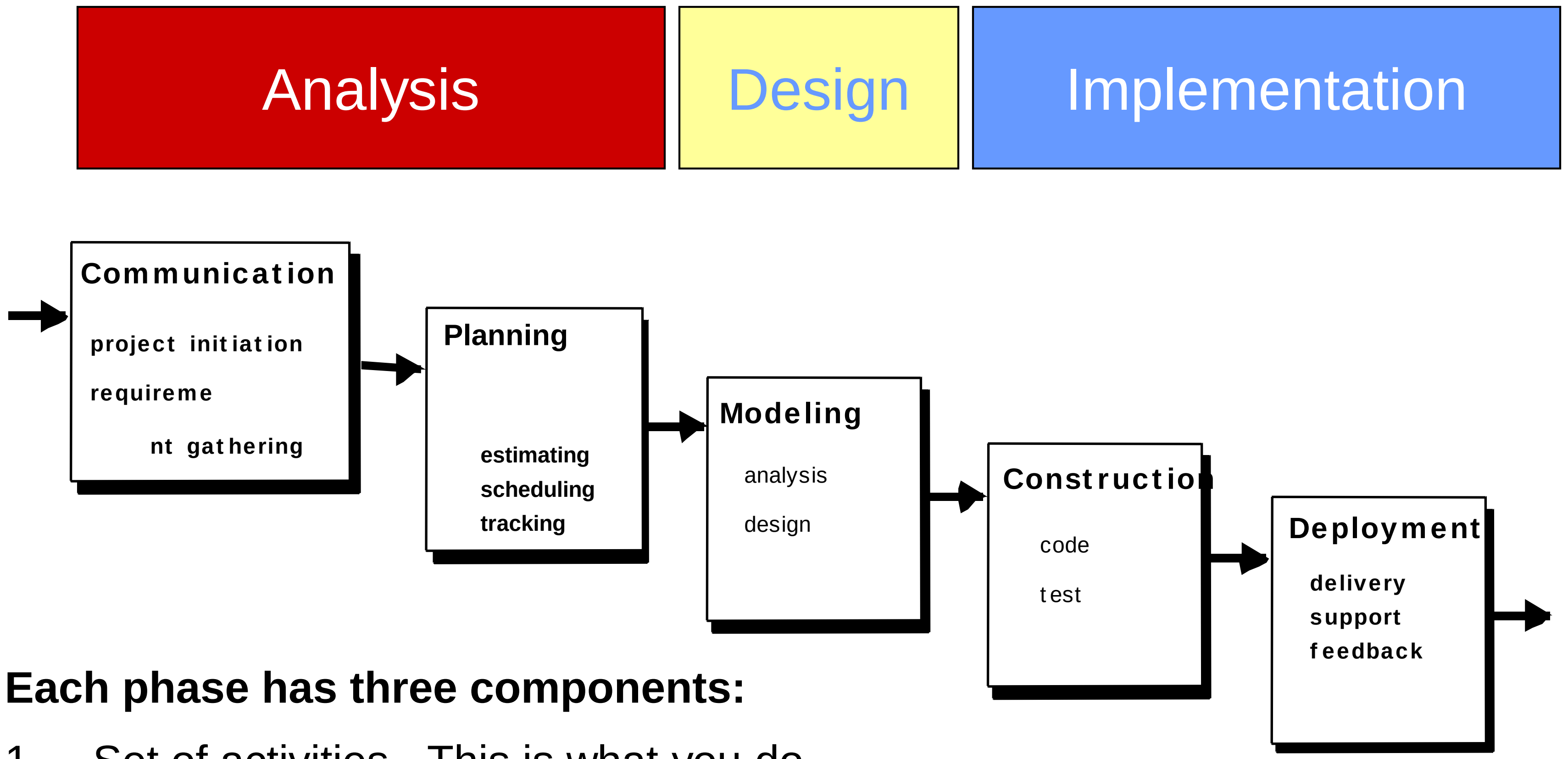
SE Activities



Each phase has three components:

1. Set of activities - This is what you do
2. Set of deliverables - This is what you produce.
3. Quality Control Measures - This is what you use to evaluate the deliverables.

Introduction

- OO

Objects

- Are entities in a computer program that have three characteristics
  - State (determined by the data fields or attributes)
  - Behavior (determined by the methods or operations of the object)
  - Identity (2 students objects, each has its own identity)
- Voice Mail box object states
  - Empty
  - Full
  - These states impact the object response to certain events.
    - When **Full** play a message indicating so …
- Bounded Buffer as an implementation of the voice mail box object

Object-oriented Programming (OOP)

- Class vs. Object
- Support OO concepts
  - Inheritance
  - Polymorphism
  - Encapsulation
  - Abstraction
- Reusable components
- Etc.

Classes and Objects Examples

- Date class
  - data: month, day, year
  - operations to set and return month, day, year
  - a Date object
    - June
    - 23
    - 2004
- *Student* class
  - Data: name, year, and grade point average
  - Methods: store/get the value of each piece of data, promote to next year, etc.
  - *Student* Object: student1
    - Data: Maria Gonzales, Sophomore, 3.5

Classes and Objects

- An object:
  - Point point1 = new Point();
- Attributes or fields:

  int x, y;
- A method:

  void move(int dx, int dy)
- A message:

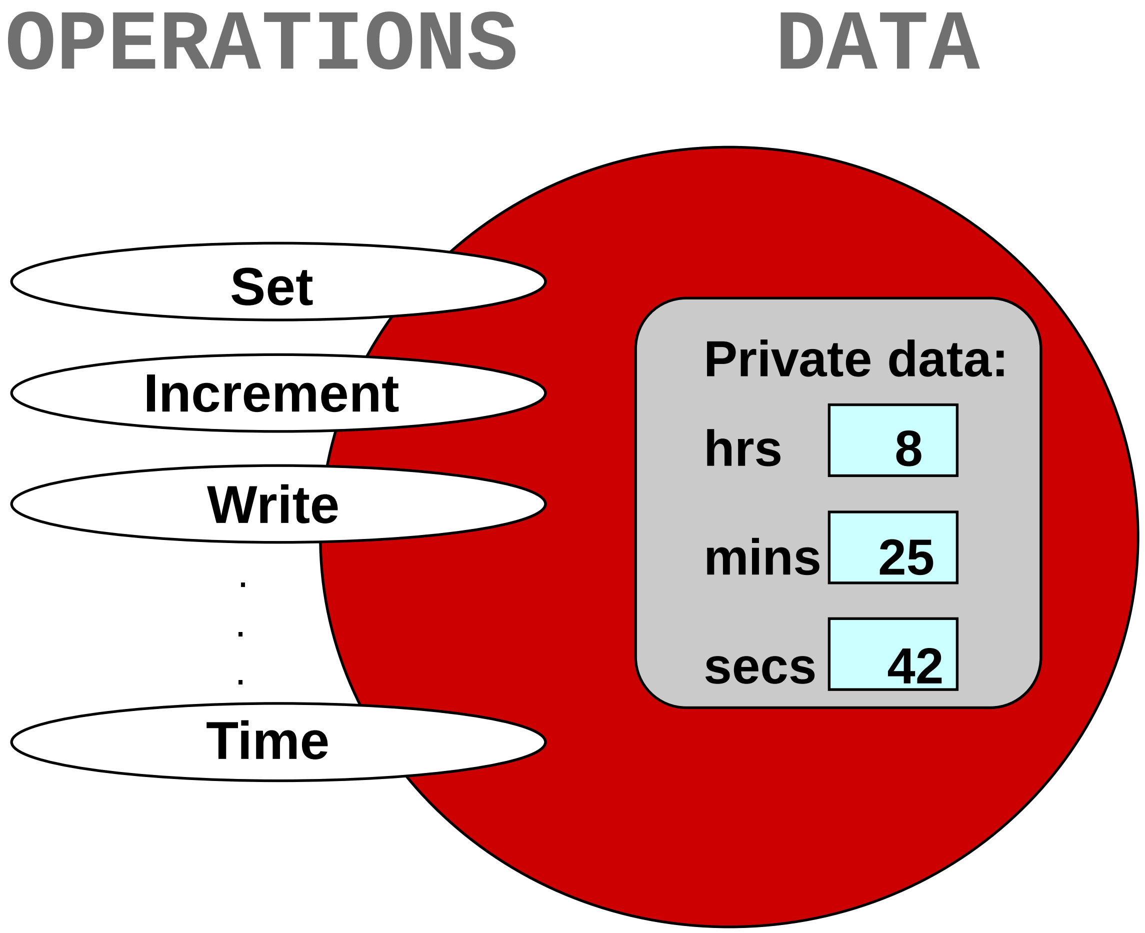
  point1.move(10, 10)

```
Class Point {

  int x, y;

  public void move (int dx, int dy) {
          x +=dx;
          y +=dy;

   }
}
```

OOP Advantage: Reuse

- Well-written classes can be reused in new applications
- Shortens **development time** because programmers don't need to write new code
- Programs are more **robust** because the class code is already tested

An Object of `class Time`

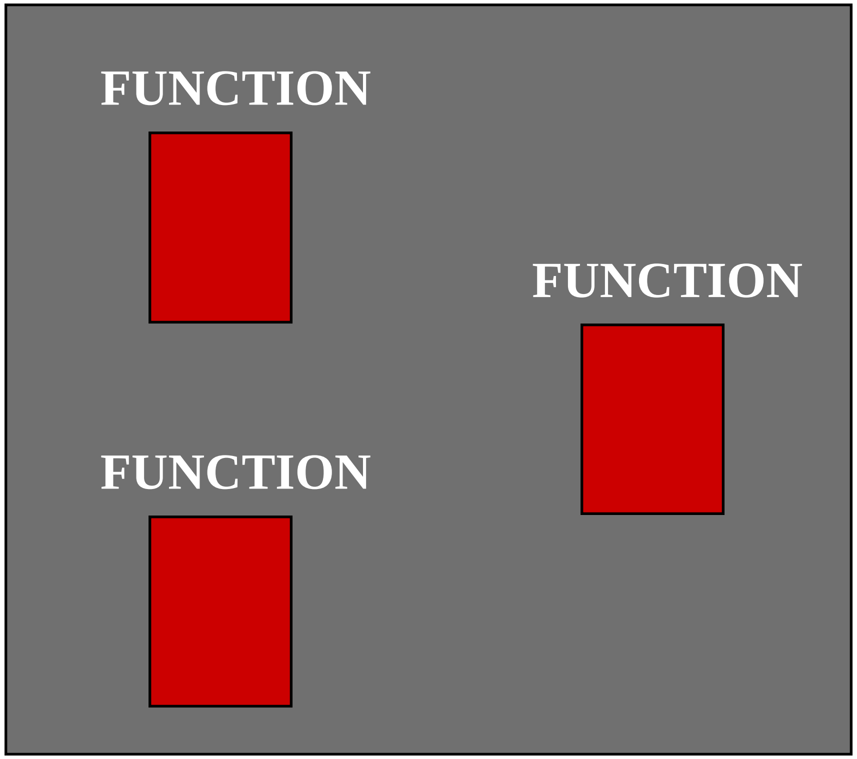


OO Design

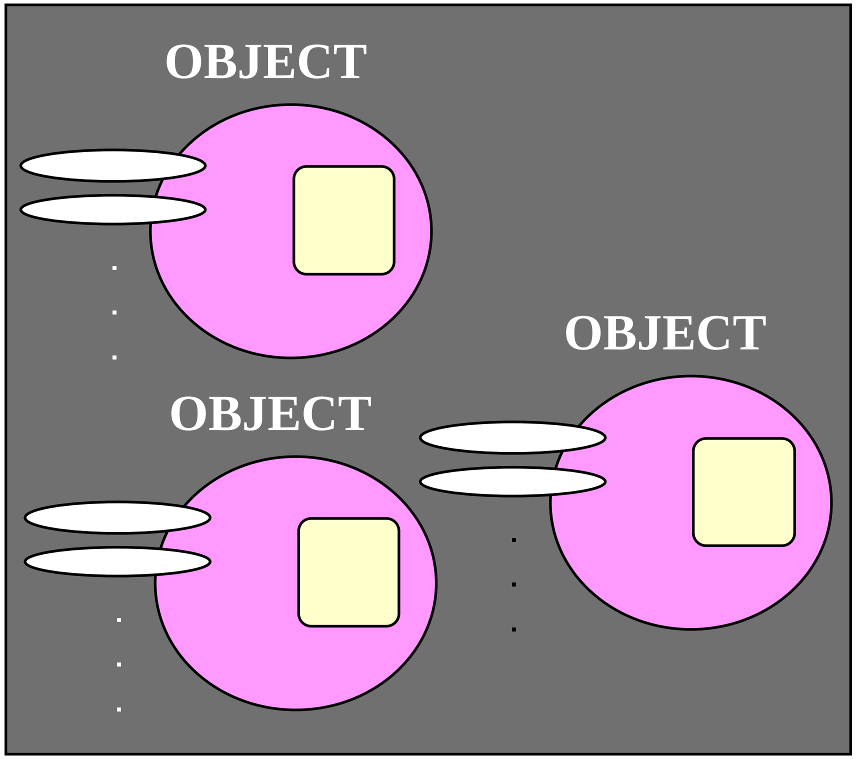
- OO Concepts

Two Design Strategies

FUNCTIONAL DECOMPOSITION



OBJECT-ORIENTED DESIGN



Object-Oriented Programming

- Classes combine <u>data</u> and the <u>methods</u> (code) to manipulate the data

- Classes are a template used to create specific objects

- All Java programs consist of at least one class.

Some Terminology

- **Object reference**: identifier of the object
- **Instantiating an object**: creating an object of a class
- **Instance of the class**: the object
- **Methods:** the code to manipulate the object data
- **Calling a method**: invoking a service for an object.

Class Data

- **Instance variables**: variables defined in the class and given values in the object
- **Fields**: instance variables and *static* variables (we'll define *static* later)
- **Members** of a class: the class's fields and methods
- Fields can be:
  - any primitive data type (*int*, *double*, etc.)
  - objects

Encapsulation

- Implementation of a module should be separated from its users.
- Instance variables are usually declared to be ***private***, which means users of the class must reference the data of an object by calling methods of the class.
- Thus the methods provide a protective shell around the data. We call this **encapsulation**.
- Benefit: the class methods can ensure that the object data is always valid.

Modularity

- Divide-and-conquer

- Decompose a complex system into a highly cohesive loosely coupled modules.
  - Cohesion single-mind-ness of a module
    - Each module should small and simple
  - Coupling the degree of connectivity between modules
    - Interactions between modules should be simple

Abstraction

- Separating the essential from the nonessential
- The behaviors of a module should be characterized in a precise way using its interfaces.
- Server provider for example is a client that provide services (methods) to it’s clients.
  - Clients need to know only the contract API.
- Abstraction
  - Denotes the essential characteristics of an object that distinguishes it from all kinds of objects.
    - What does the object do without any implication on how does it do it
  - A software object is an abstraction
    - A representation of something in the real word like a student, a book, …

Principals of OO - Encapsulation

- Encapsulation
  - The process of Compartmentalizing the elements of an object that constitute its structure and behavior
    - Data hiding
    - Localize design decisions
  - What are the encapsulated elements?
    - **Information** that describes the object (the things an object know about itself)
      - No of pages, cover type, ISBN, and other Book information
      - State, the object’s current condition (or state)
    - **Behavior**
      - What the object can do (register, drop etc.)
      - What can be done to it (a pencil object can Write!!)

Encapsulation

Polymorphism

- Polymorphism - Many shapes
  - The same operation may behave differently on different classes.
    - Example + sign
    - Same method many ways to call it (dynamic)
  - Overloading of operations
    - Same operation with different signatures
  - When an object is substituted with one of its children at run time
    - (The is-a relationship)

Reusability

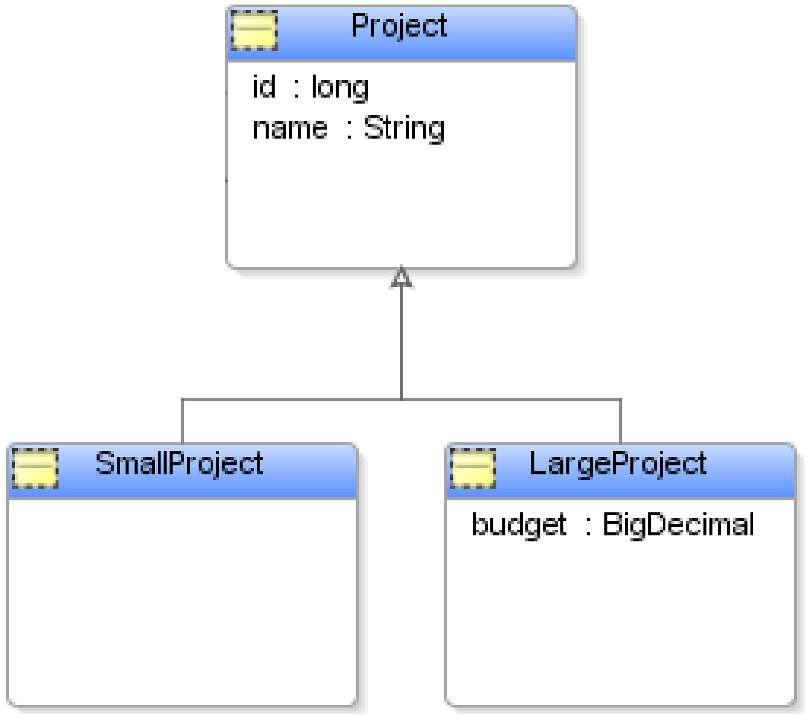
- **Reuse**: class code is already written and tested, so you build a new application faster and it is more reliable

Example: A *Date* class could be used in a calendar program, appointment-scheduling program, online shopping program, etc.

How To Reuse A Class

- You don't need to know how the class is written.
- You do need to know the **application programming interface** (**API**) of the class.
- The API is published and tells you:
  - How to create objects
  - What methods are available
  - How to call the methods

Inheritance



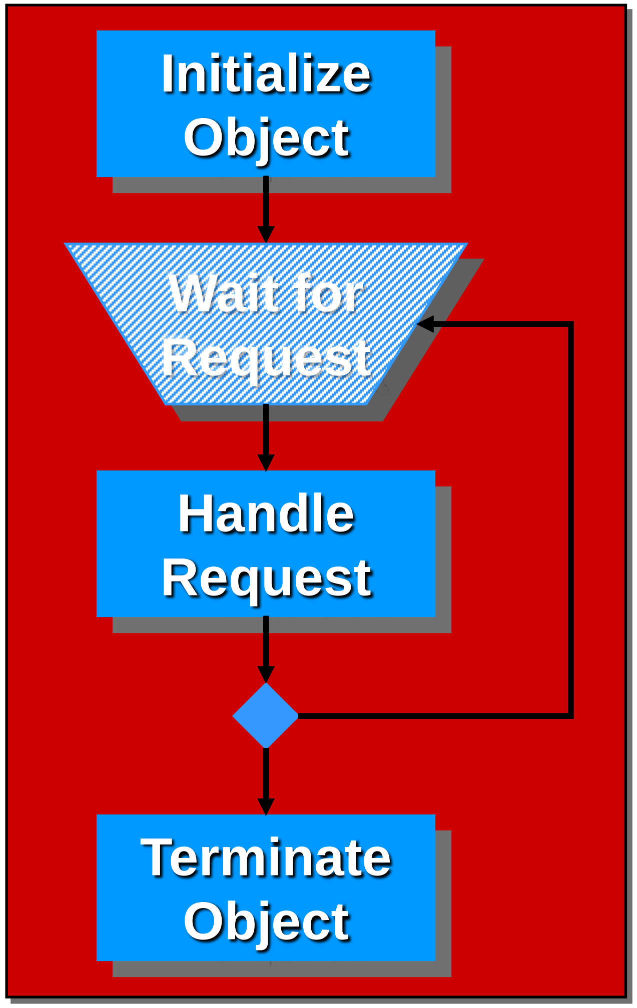
- Wiki

"In object-oriented programming, **inheritance** is a way to form new classes (instances of which are called objects) using classes that have already been defined. The inheritance concept was invented in 1967 for Simula.

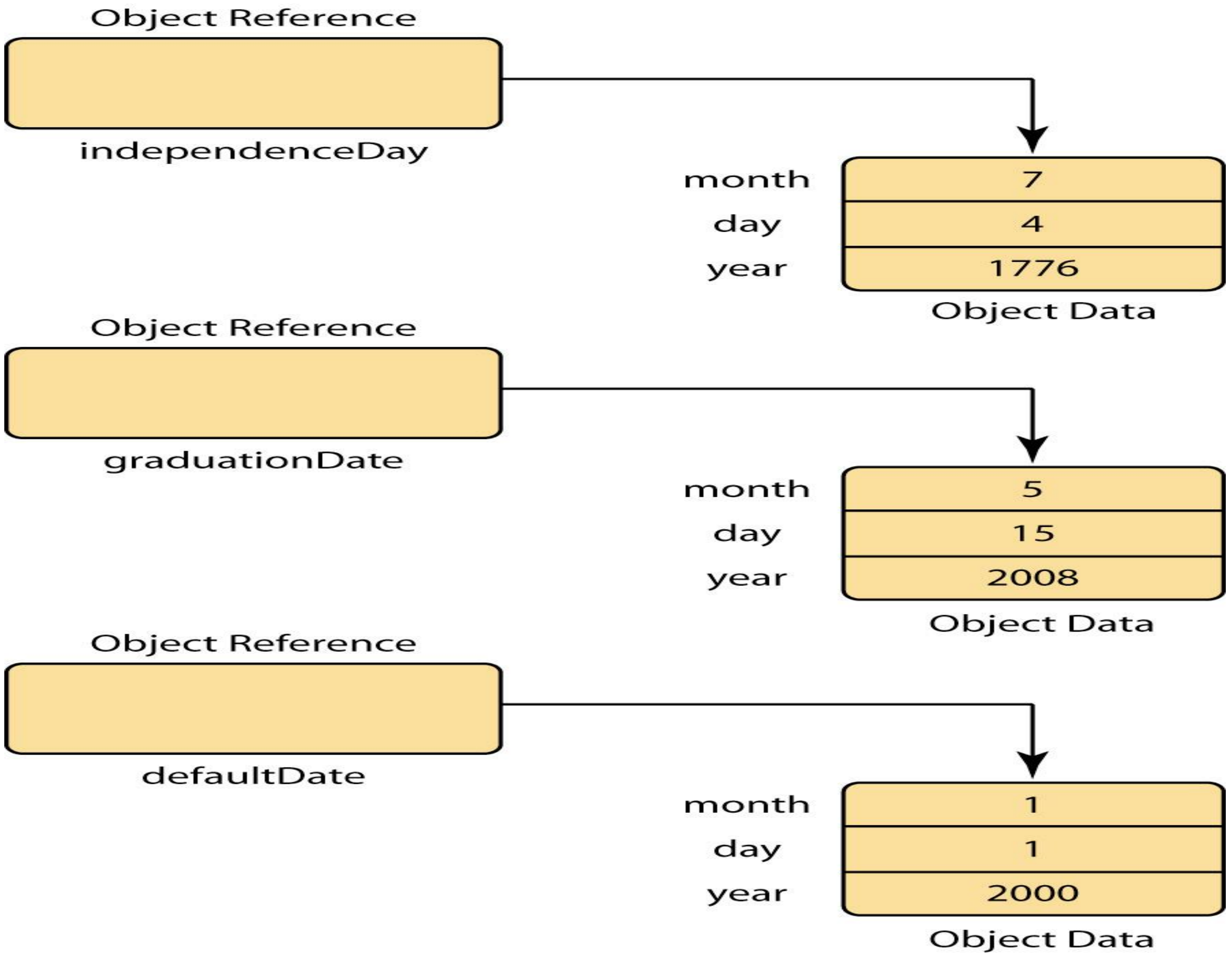
The new classes, known as **derived classes**, take over (or **inherit**) attributes and behavior of the pre-existing classes, which are referred to as **base classes** (or ancestor classes). It is intended to help reuse existing code with little or no modification.

"

Object Behavior - **Object Life Cycle**

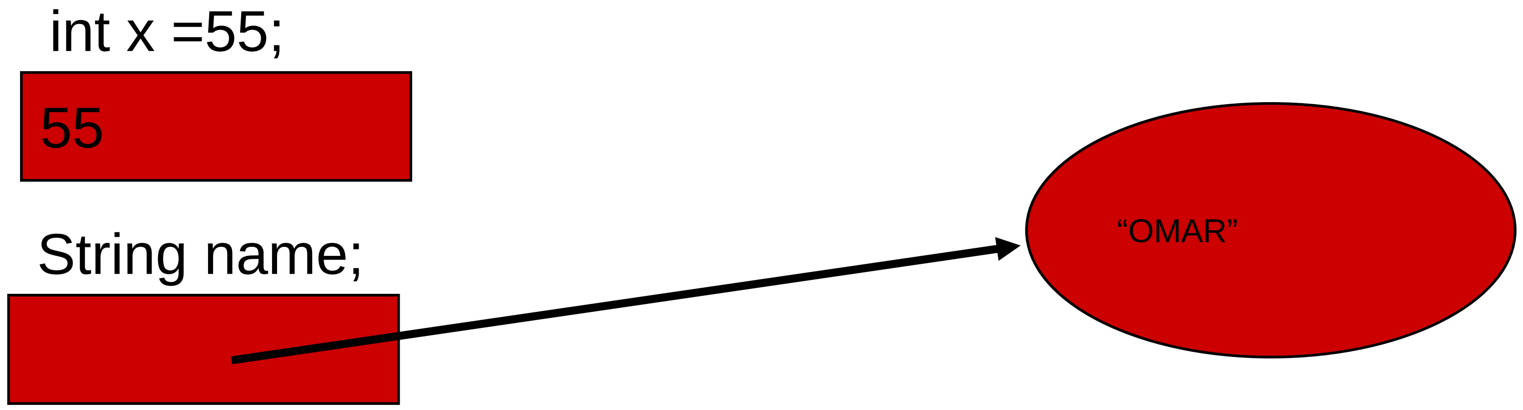


Objects After Instantiation



Notes

- Public methods of a class denote services objects of the class can perform
- Only primitive types in Java store their value, all other types the variable stores a reference to the data



Object Reference vs. Object Data

- Object references point to the location of object data.

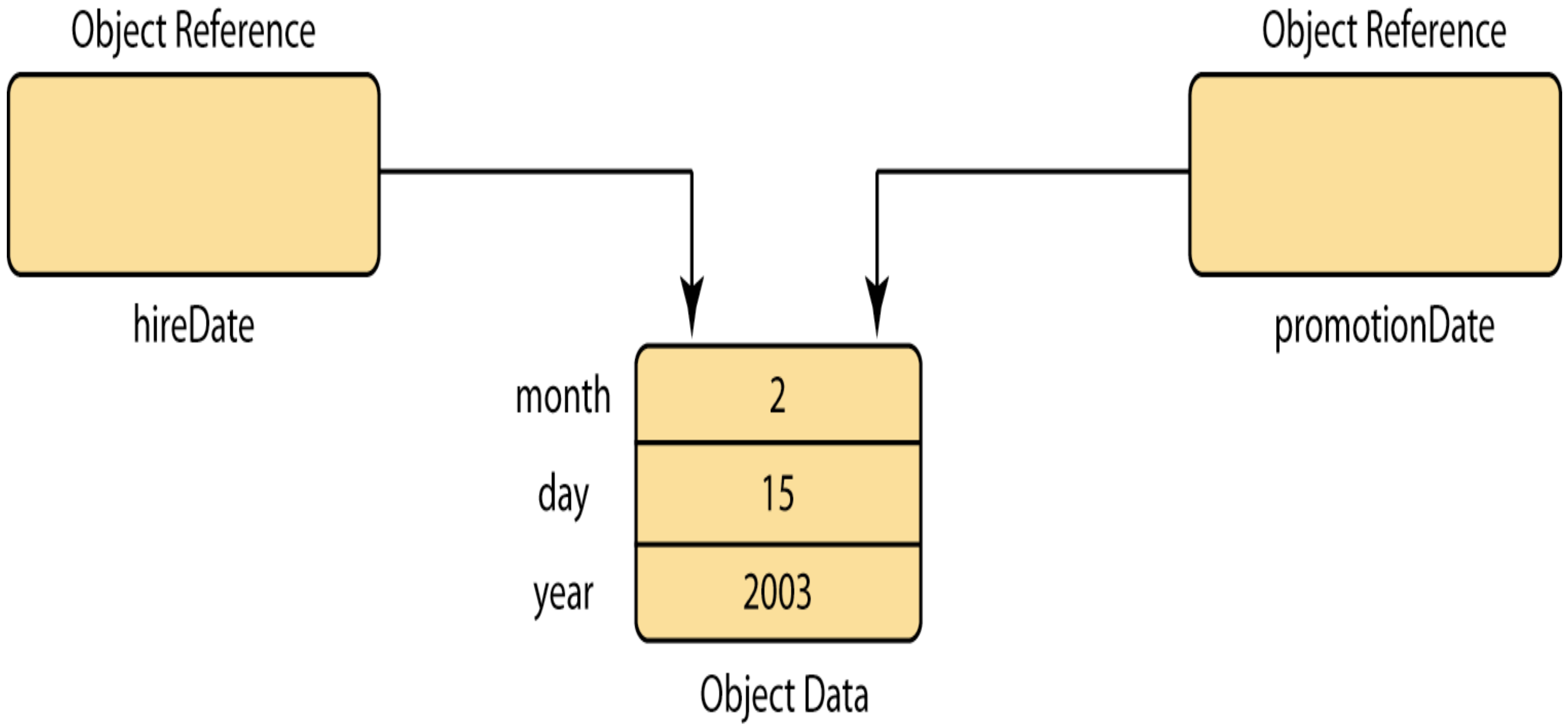
- An object can have multiple object references pointing to it.
  - Or an object can have no object references pointing to it.
  - If so, the **garbage collector** will free the object's memory
- *See*
  - Example Next Slide

Example ObjectReferenceAssignment.java

```
public class ObjectReferenceAssignment {
  public static void main( String [] args )   {
     Date hireDate = new Date( 2, 15, 2003 );
     System.out.println( "hireDate is " + hireDate.getMonth( )  + "/" +
                              hireDate.getDay( )  + "/" + hireDate.getYear( ) );
     Date promotionDate = new Date( 9, 28, 2004 );
     System.out.println( "promotionDate is " + promotionDate.getMonth( )
                + "/" + promotionDate.getDay( ) + "/" + promotionDate.getYear(
       ) );
     promotionDate = hireDate; 🡐
     System.out.println( "\nAfter assigning hireDate “  + "to promotionDate:"
       );
     System.out.println( "hireDate is " + hireDate.getMonth( )
                   + "/" + hireDate.getDay( )  + "/" + hireDate.getYear( ) );
     System.out.println( "promotionDate is " + promotionDate.getMonth( ) +
       "/" +
                promotionDate.getDay( ) + "/" + promotionDate.getYear( ) );
  }
}
```

Two References to an Object

- After the example runs, two object references point to the same object



null Object References

- An object reference can point to no object. In that case, the object reference has the value *null*

- Object references have the value *null* when they have been declared, but have not been used to instantiate an object.

- Attempting to use a *null* object reference causes a *NullPointerException* at run time.

Example NullReference.java

```
public class NullReference
{
  public static void main( String[] args )
  {
     Date aDate;
     aDate.setMonth( 5 );
  }
}
```

static Methods

- Also called **class methods**
- Can be called <u>without</u> instantiating an object
- Might provide some quick, one-time functionality,
  - for example, popping up a dialog box
  - Math.abs(-9);
- Stateless Objects
- In method API, keyword *static* precedes return type

static dataType mthodName (arg1,ard2,…);

Calling *static* Methods

- Use dot syntax with **class name** instead of object reference
- Syntax:

```
ClassName.methodName( args )
```

- Example:

```
int absValue = Math.abs( -9 );
```

- Uses of class methods
  - Provide access to class variables without using an object

static Class Variables

- Class variables are variables that are stored among all objects of a given class
  - Defining constants
    - `public final static c=299792458`
  - Allow all objects of a class to share a piece of data
    - How many objects have been created
- Syntax:

```
ClassName.staticVariable
```

- Example:

```
Color.BLUE
```

BLUE is a *static* constant of the *Color* class.

Static Class Variables and Static Methods

```
class Counter {
  private int value;
  private static int  numCounters = 0; 🡐
  public Counter()   {
    value = 0;
    numCounters++;
  }
public static int getNumCounters() { 🡐
return numCounters; }
}
...
System.out.println("Number of counters: "
                   + Counter.getNumCounters()); 🡐
```

Static vs. Instance Variable

- Class (static)
  - the class has one copy for all instances
  - Can use class variables In instance methods and in class methods

- instance variables
  - each instance has its own copy
  - Can use instance variables in instance methods only