Basic OOP Concepts and Terms

In this class, we will cover:

- Objects and examples of different object types
- Classes and how they relate to objects
- Object attributes and methods
- Object interactions and messages
- Encapsulation and information hiding

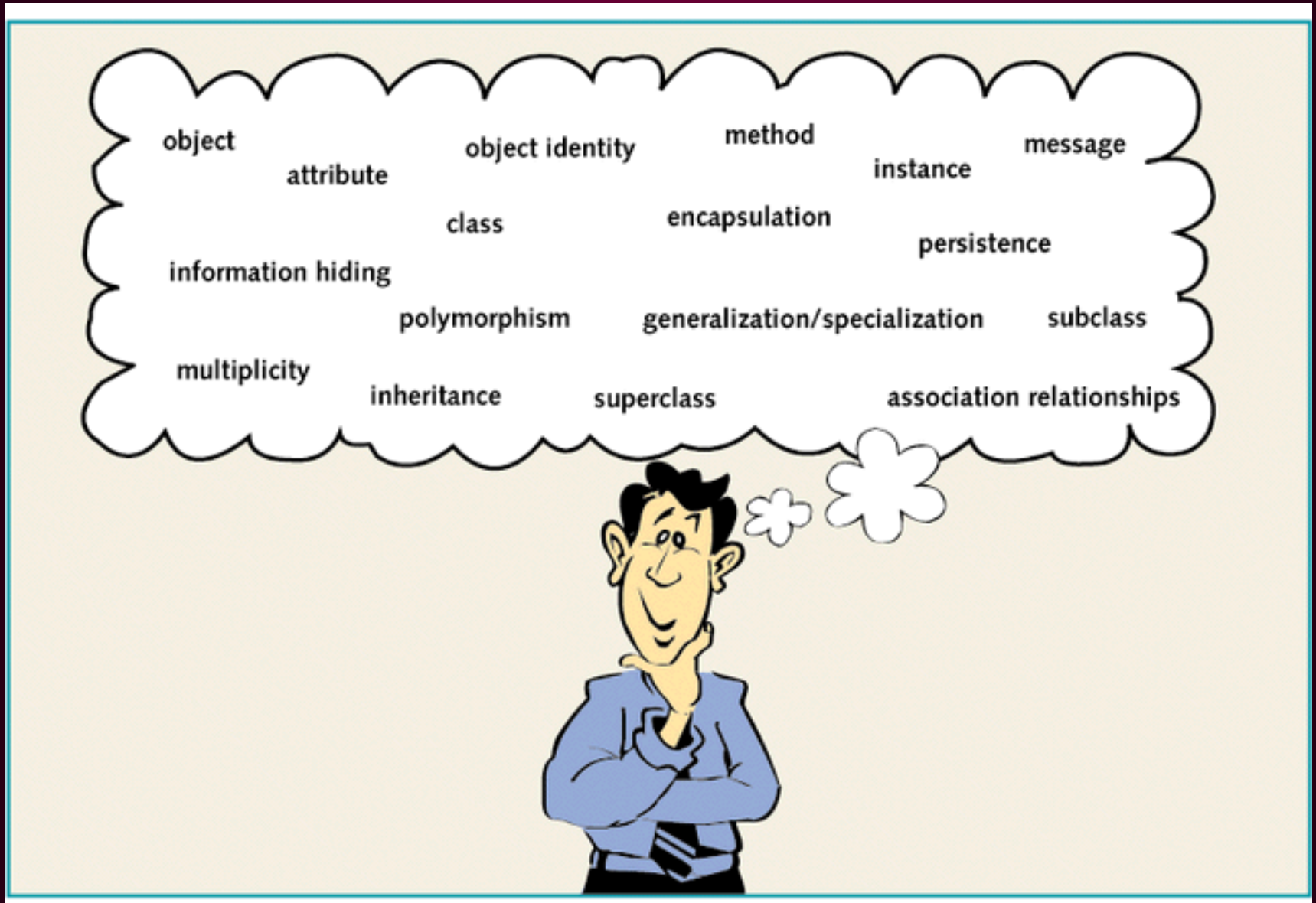


Figure 1-5 Key OO concepts

Objects

- Most basic component of OO design. Objects are designed to do a *small*, specific piece of work.
- Objects represent the various components of a business system

Examples of Different Object Types

- GUI objects
  - objects that make up the user interface
  - e.g. buttons, labels, windows, etc.

- Problem Domain objects
  - Objects that represent a business application
  - A problem domain is the scope of what the system to be built will solve. It is the business application.
    - e.g. An order-entry system
      A payroll system
      A student system

Sample Problem Domain

Company ABC needs to implement an order-entry system that takes orders from customers for a variety of products.

What are the objects in this problem domain?

Hint: Objects are usually described as nouns.

Possible Objects for this Problem Domain

Customer

Order

Product

Classes and Objects

- **Classes**
  - **Define what all objects of the class represent**
  - **It is like a blueprint. It describes what the objects look like**
  - **They are a way for programs to model the real world**
- **Objects**
  - **Are the instances of the class**

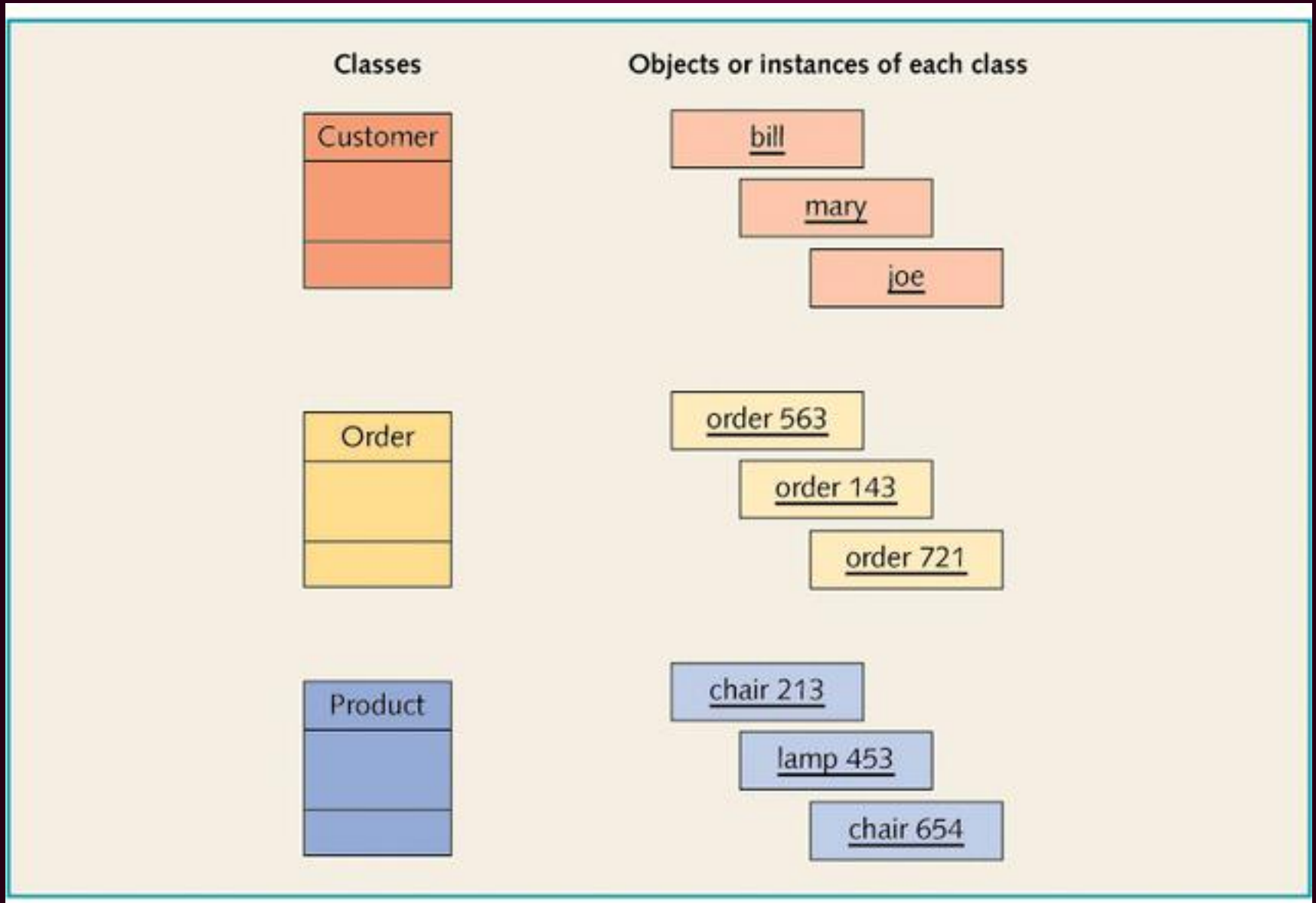


Figure 1-9 Class versus objects or instances of the class

Object Attributes and Methods

- **Classes contain two things:**
  - **Fields** (attributes, data members, class variables):
    - Data items that differentiate one object of the class from another. e.g. employee name, student number
    - Characteristics of an object that have values
    - What are some possible attributes for the customer object?
  - **Methods** (behaviors):
    - Named, self-contained blocks of code that typically operate on the fields
    - Describe what an object can do
    - Can be thought of as the verbs in a problem domain
    - What are some possible methods for the customer object?

Problem Domain Objects	Attributes	Methods
Customer	name, address, phone number	set name, set address, add new order for customer
Order	order number, date, amount	set order date, calculate order, amount, add product to order, schedule order shipment
Product	product number, description, price	add to order, set description, get price

Figure 1-7 Attributes and methods in problem domain objects

GUI Objects	Attributes	Methods
Button	size, shape, color, location, caption	click, enable, disable, hide, show
Label	size, shape, color, location, text	set text, get text, hide, show
Form	width, height, border style, background color	change size, minimize, maximize, appear, disappear

Figure 1-6 Attributes and methods of GUI objects

Object Interactions and Messages

- **Objects interact with other objects in a variety of relationships**
  - e.g. one-to-one, one-to-many

- **Messages**
  - The means by which objects interact
  - Example:
    - User initiates interaction via messages to GUI objects
    - GUI objects interact with problem domain objects via messages
    - Problem domain objects interact with each other and GUI objects via messages
    - GUI objects respond to user via messages

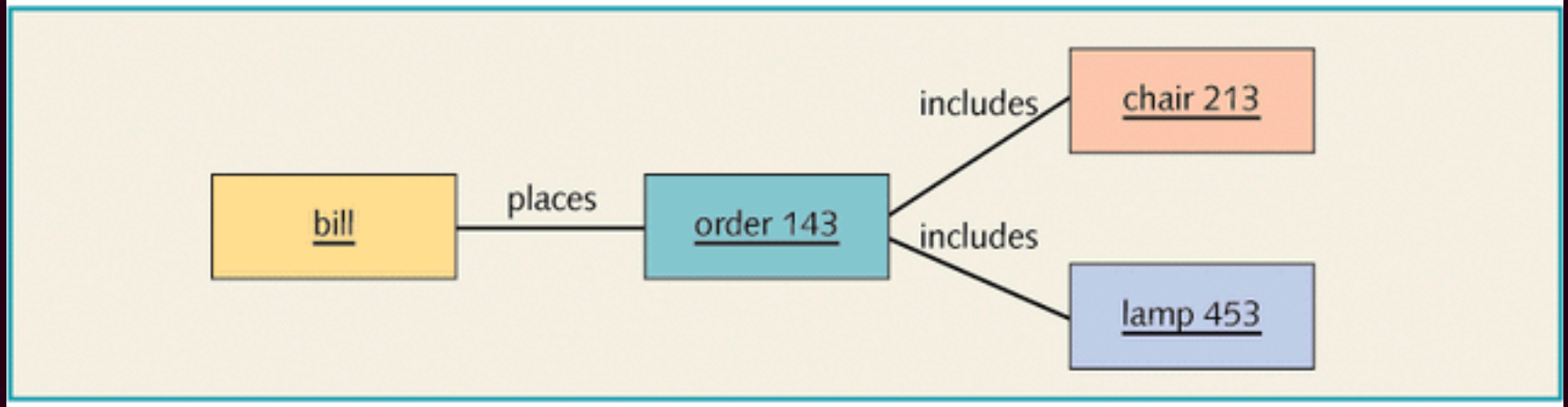


Figure 1-10 Associating objects with other objects

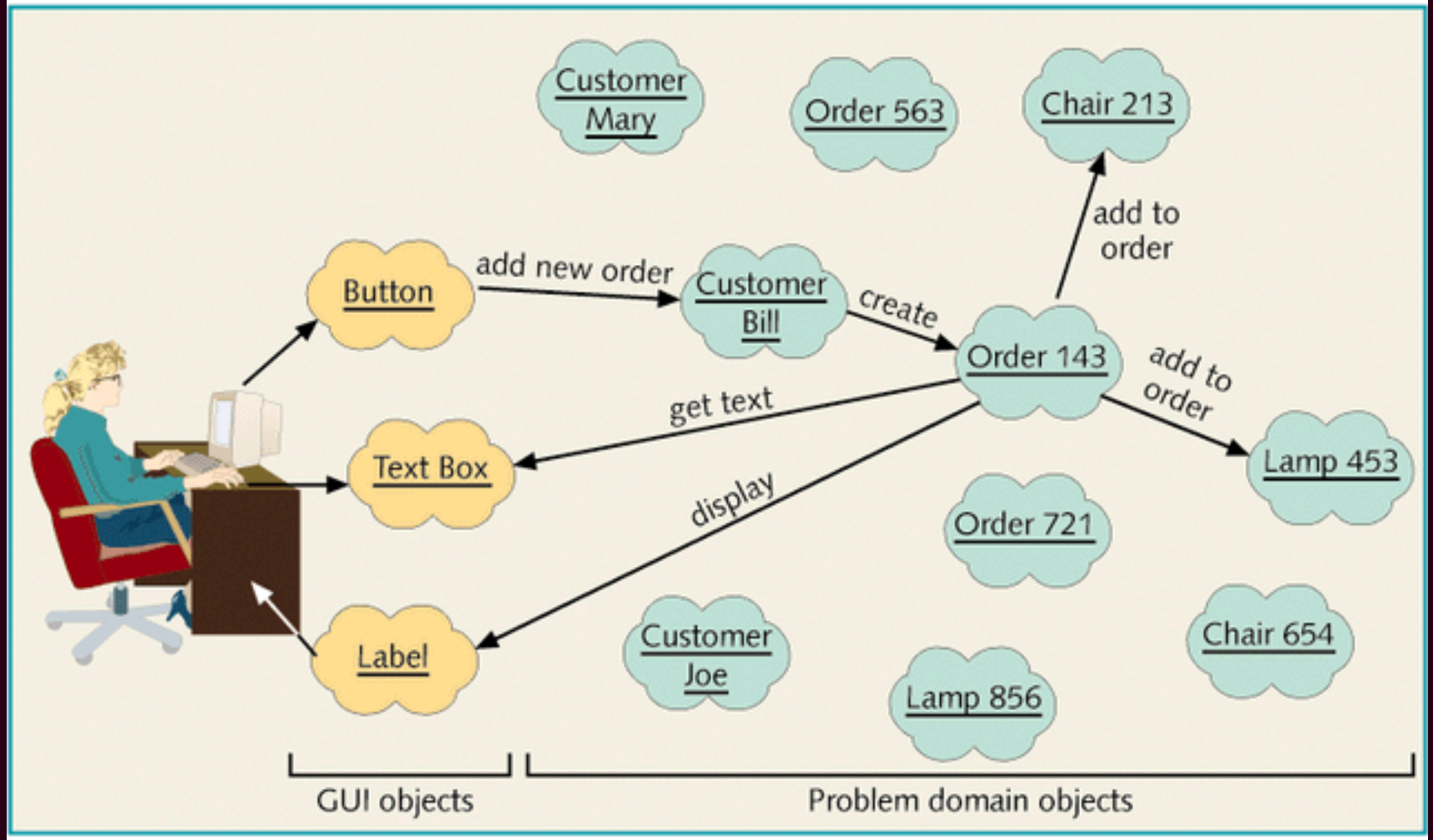


Figure 1-8 Order-processing system where objects interact by sending messages

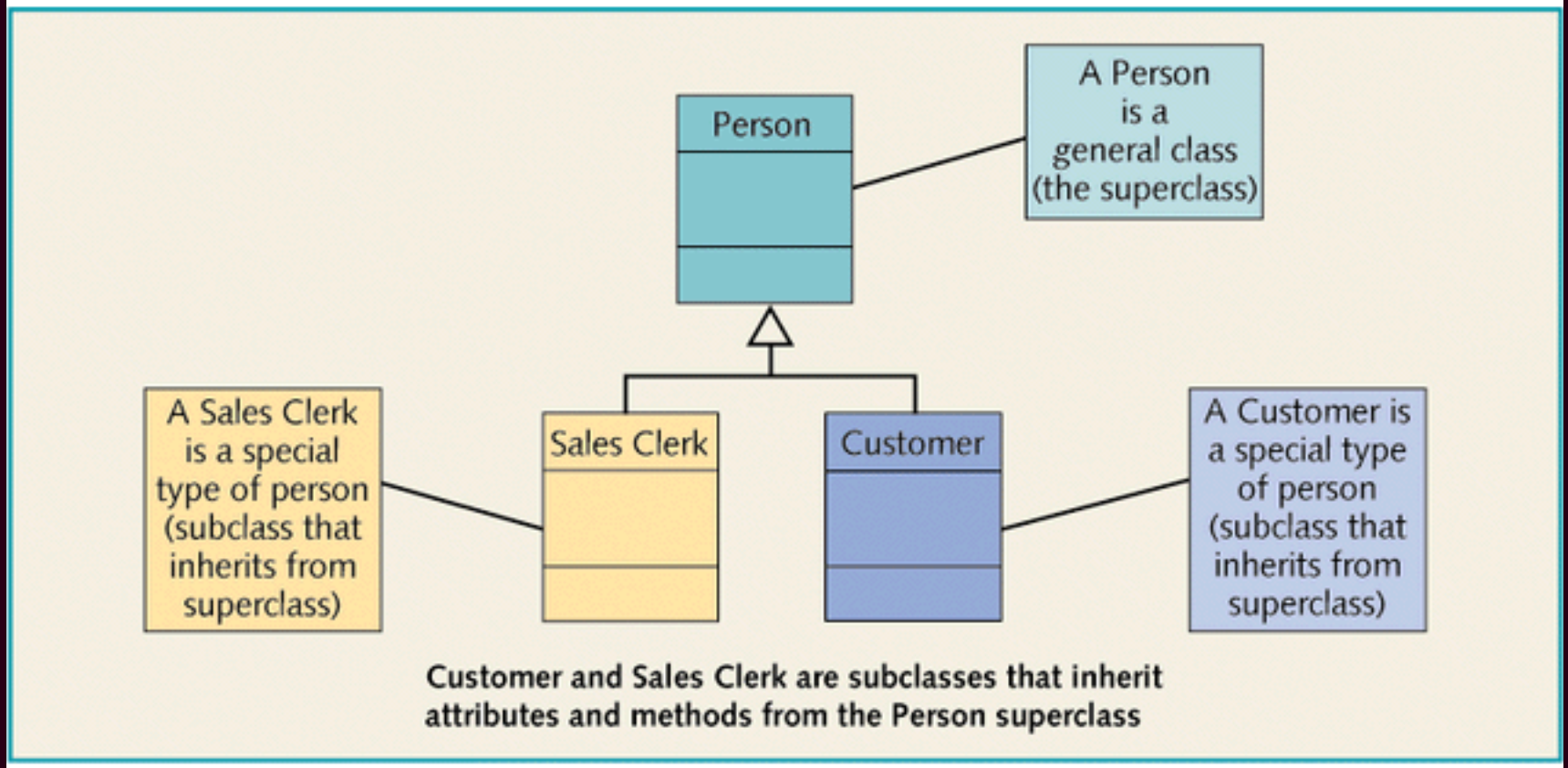
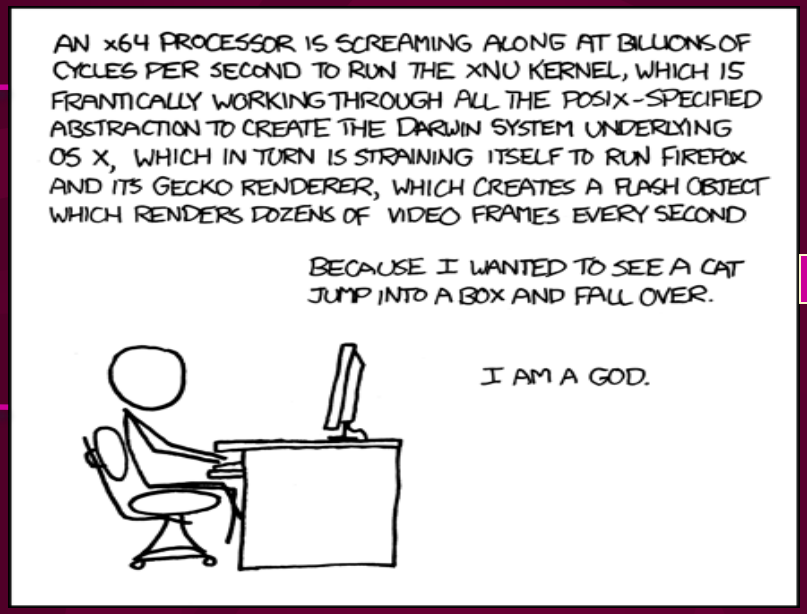


Figure 1-11 Superclass and subclass

Encapsulation and Information Hiding

Don't need to know this



Can focus on this!!

Encapsulation and Information Hiding

- **Encapsulation**
  - Objects have attributes and methods combined into one unit
- **Information Hiding**
  - Hiding the internal structure of objects, protecting them from corruption
- **Identity**
  - Unique reference for each object
- **Persistent objects**
  - Defined as available for use over time

Encapsulation and Information Hiding

- A field can be declared private.
- No code outside the class can access or change it.

  private ***type name****;*
- Examples:

  private int id;

  private String name;
- Client code sees an error when accessing private fields:
- PointMain.java:11: x has private access in Point
- System.out.println("p1 is (" + p1.x + ", " + p1.y + ")");

Encapsulation and Information Hiding

- We can provide methods to get and/or set a field's value:

```
// A "read-only" access to the x field ("accessor")
public int getX() {
    return x;
}
// Allows clients to change the x field ("mutator")
public void setX(int newX) {
    x = newX;
}
```

- Client code will look more like this:

```
System.out.println("p1: (" + p1.getX() + ", " + p1.getY() + ")");
p1.setX(14);
```

Encapsulation and Information Hiding
Point Class

```
// A Point object represents an (x, y) location.
public class Point {
    private int x;
    private int y;

    public Point(int initialX, int initialY) {
        x = initialX;
        y = initialY;
    }

    public double distanceFromOrigin() {
        return Math.sqrt(x * x + y * y);
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    public void setLocation(int newX, int newY) {
        x = newX;
        y = newY;
    }

    public void translate(int dx, int dy) {
        x = x + dx;
        y = y + dy;
    }
}
```

Encapsulation and Information Hiding Client Code

```
public class PointMain4 {
   public static void main(String[] args) {
       // create two Point objects
       Point p1 = new Point(5, 2);
       Point p2 = new Point(4, 3);

       // print each point
       System.out.println("p1: (" + p1.getX() + ", " + p1.getY() +
  ")");
       System.out.println("p2: (" + p2.getX() + ", " + p2.getY() +
  ")");

       // move p2 and then print it again
       p2.translate(2, 4);
       System.out.println("p2: (" + p2.getX() + ", " + p2.getY() +
  ")");
  }
}
```

OUTPUT:

```
p1 is (5, 2)
p2 is (4, 3)
p2 is (6, 7)
```

Encapsulation and Information Hiding
Client Code

- Provides abstraction between an object and its clients.
- Protects an object from unwanted access by clients.
  A bank app forbids a client to change an Account's balance.
- Allows you to change the class implementation.
  Point could be rewritten to use polar coordinates (radius r, angle θ), but with the same methods.
- Allows you to constrain objects' state (invariants).
  Example: Only allow Points with non-negative coordinates.