Bulgarian Association
Of Software Developers

Object-Oriented Programming Concepts

Contents

1. What is OOP?
2. Classes and Objects
3. Principles of OOP
   - Inheritance
   - Abstraction
   - Encapsulation
   - Polymorphism

What is OOP?

What is OOP?

- Object-oriented programming (OOP) is an engineering approach for building software systems
  - Based on the concepts of classes and objects that are used for modeling the real world entities
- Object-oriented programs
  - Consist of a group of cooperating objects
  - Objects exchange messages, for the purpose of achieving a common objective
  - Implemented in object-oriented languages

OOP in a Nutshell

- A program models a world of interacting objects
- Objects create other objects and “send messages” to each other (in Java, call each other’s methods)
- Each object belongs to a class
  - A class defines properties of its objects
  - The data type of an object is its class
- Programmers write classes (and reuse existing classes)

What are OOP's Claims To Fame?

- Better suited for team development
- Facilitates utilizing and creating reusable software components
- Easier GUI programming
- Easier software maintenance

- All modern languages are object-oriented: Java, C#, PHP, Perl, C++, ...

Classes and Objects

What Are Objects?

- Software objects model real-world objects or abstract concepts
  - E.g. dog, bicycle, queue
- Real-world objects have states and behaviors
  - Dogs' states: name, color, breed, hungry
  - Dogs' behaviors: barking, fetching, sleeping

What Are Objects?

- How do software objects implement real-world objects?
  - Use variables/data to implement states
  - Use methods/functions to implement behaviors
- An object is a software bundle of variables and related methods

Objects Represent

checks

people

shopping list

Things in the real world

…

numbers

characters

queues

arrays

Things in the computer world

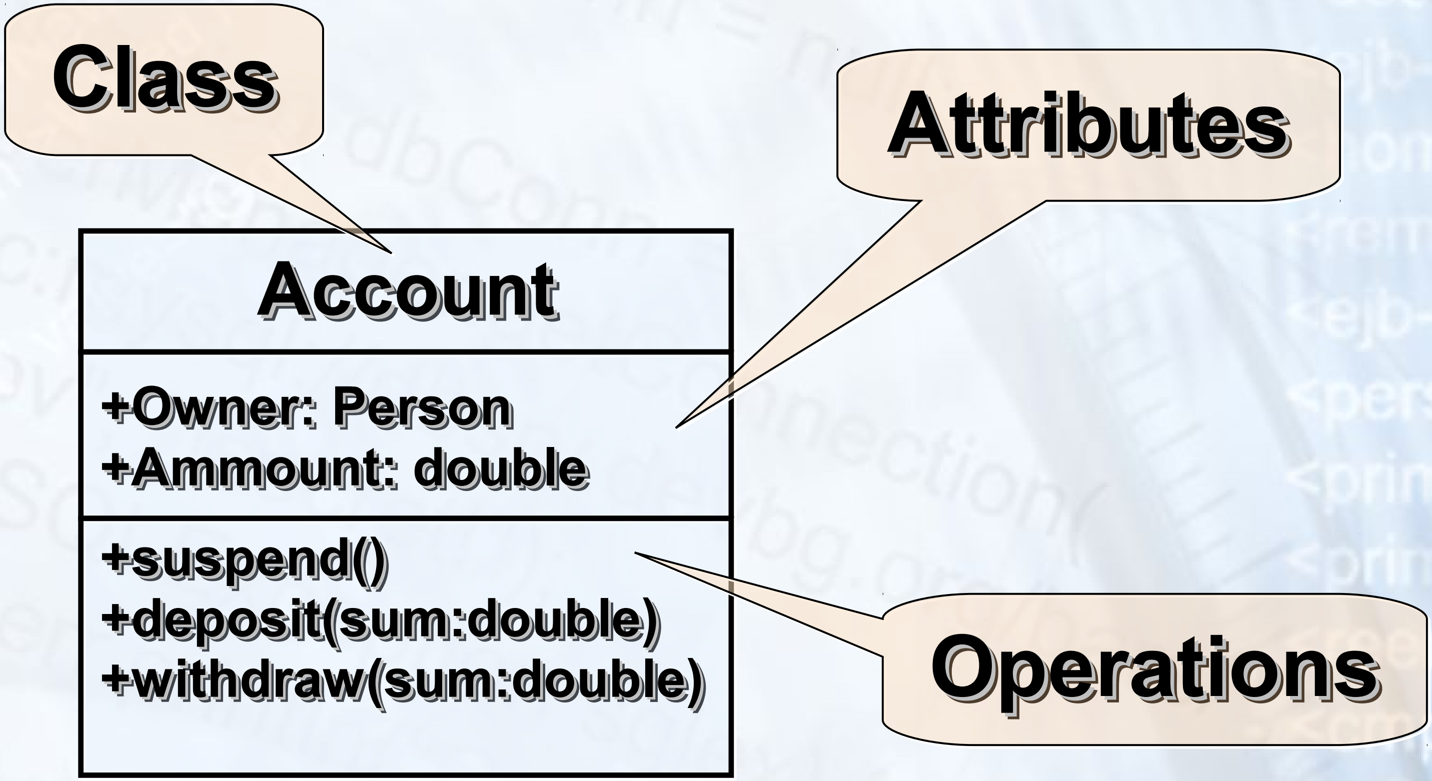
Classes

- Classes provide the structure for *objects*
  - Define their prototype
- Classes define:
  - Set of *attributes*
    - Also called *state*
    - Represented by variables and properties
  - Behavior
    - Represented by methods
- A class defines the methods and types of data associated with an object

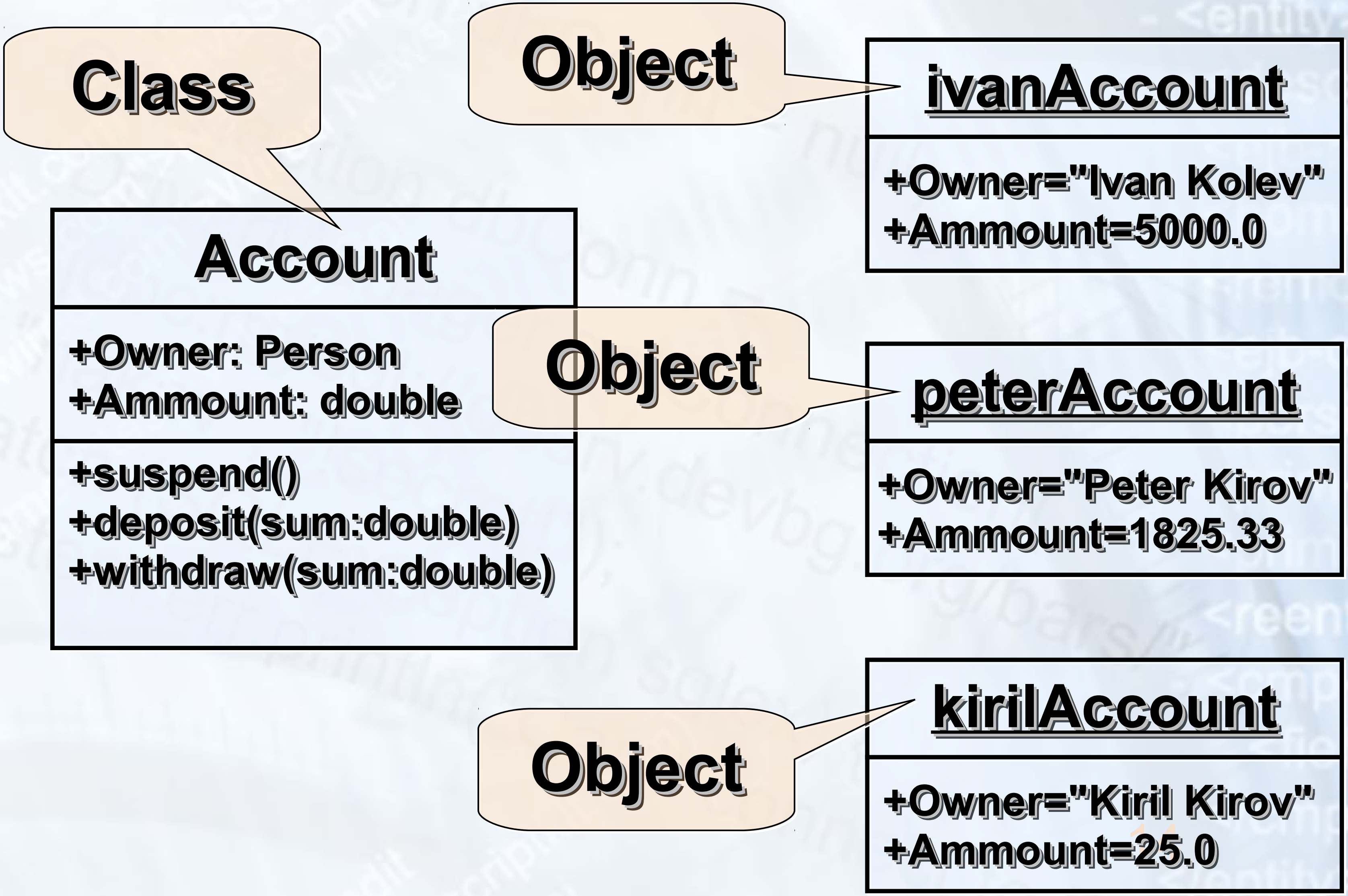
Objects

- Creating an object from a class is called *instantiation*
- An *object* is a concrete *instance* of a particular class
- Objects have state
  - Set of values associated to their attributes
- Example:
  - Class: Account
  - Objects: Ivan's account, Peter's account

Classes – Example



Classes and Objects – Example



Class

Account

+Owner: Person
+Ammount: double

+suspend()
+deposit(sum:double)
+withdraw(sum:double)

Object

ivanAccount

+Owner="Ivan Kolev"
+Ammount=5000.0

Object

peterAccount

+Owner="Peter Kirov"
+Ammount=1825.33

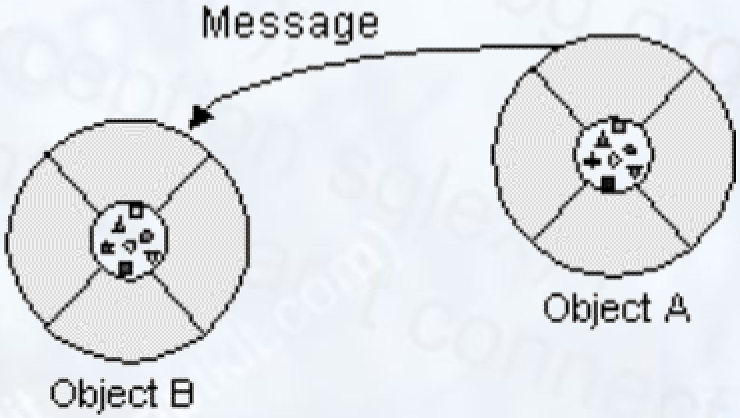
Object

kirilAccount

+Owner="Kiril Kirov"
+Ammount=25.0

Messages

- What is a message in OOP?
  - A request for an object to perform one of its operations (methods)
- All communication between objects is done via messages



Interfaces

- Messages define the interface to the object
  - Everything an object can do is represented by its message interface
- The interfaces provide abstractions
  - You shouldn't have to know anything about what is in the implementation in order to use it (black box)
- An interface is a set of operations (methods) that given object can perform

The Principles of OOP

The Principles of OOP

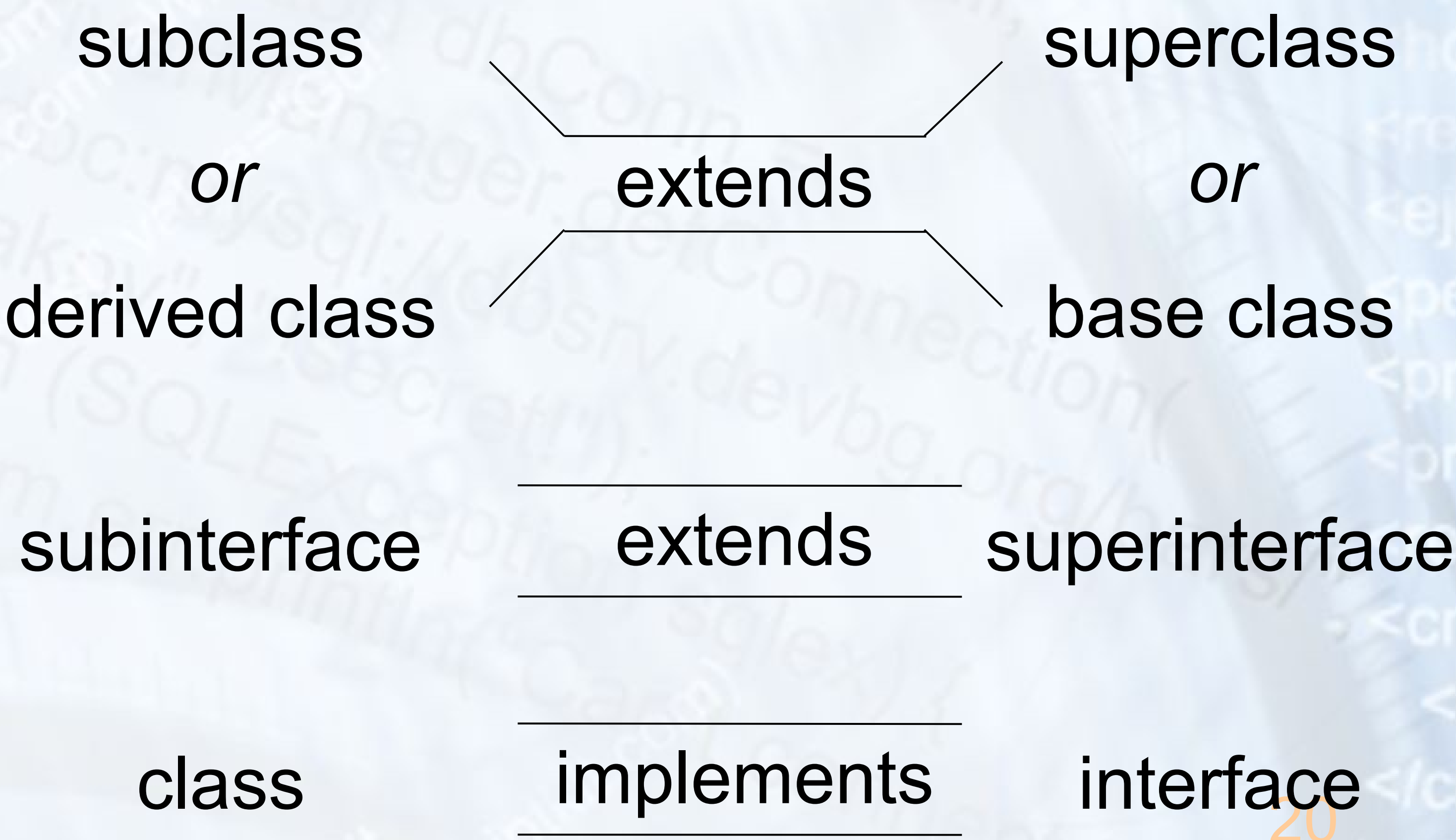
- Inheritance
- Abstraction
- Encapsulation
- Polymorphism

Inheritance

- A class can *extend* another class, inheriting all its data members and methods
  - The child class can redefine some of the parent class's members and methods and/or add its own
- A class can *implement* an interface, implementing all the specified methods
- Inheritance implements the “is a” relationship between objects

Inheritance

- **Terminology**

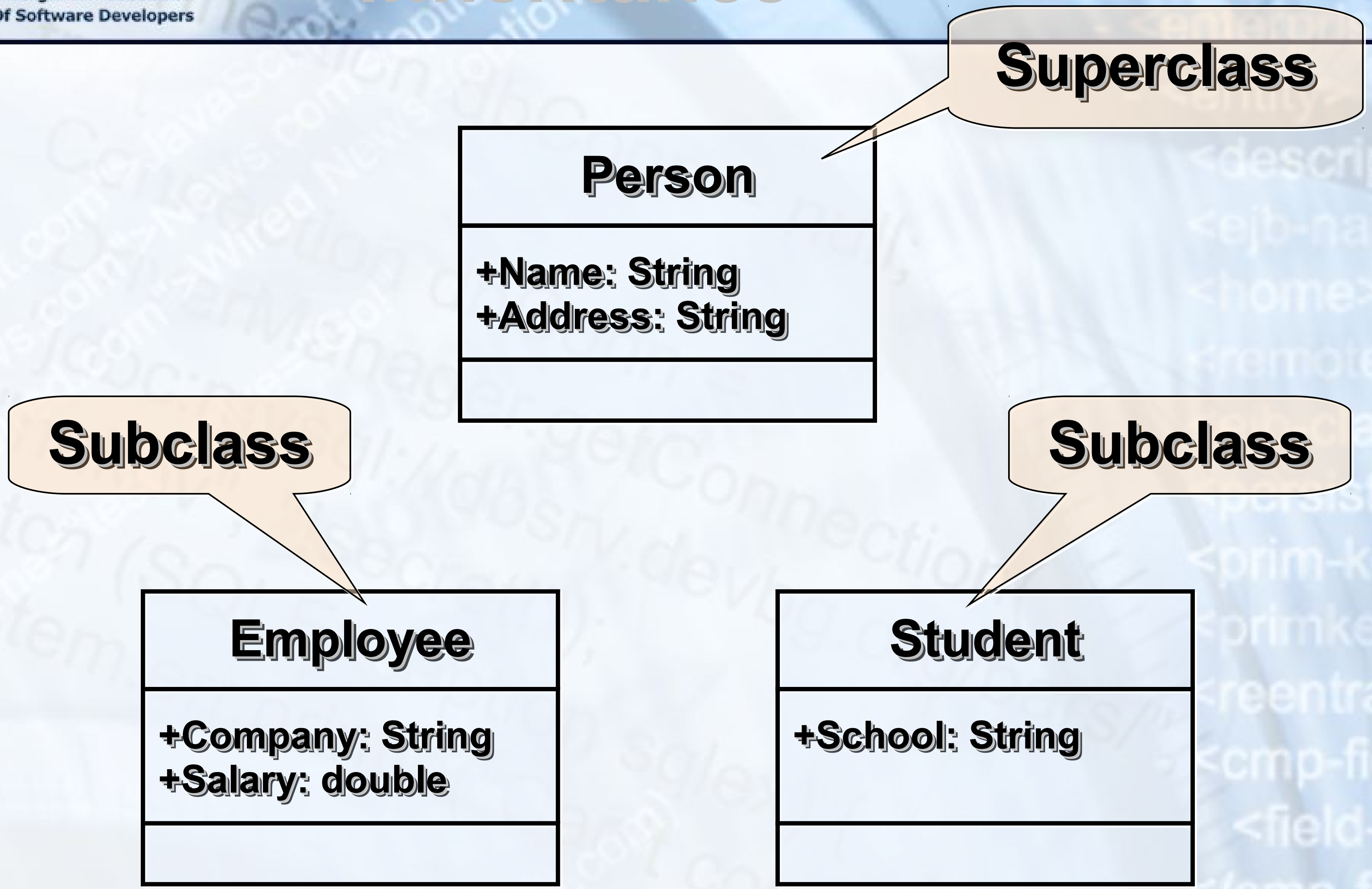


subclass *or* derived class extends superclass *or* base class

subinterface extends superinterface

class implements interface

Inheritance



Inheritance in Java

- In Java, a subclass can extend only one superclass
- In Java, a subinterface can extend one superinterface
- In Java, a class can implement several interfaces
  - This is Java’s form of *multiple inheritance*

Interfaces and Abstract Classes in Java

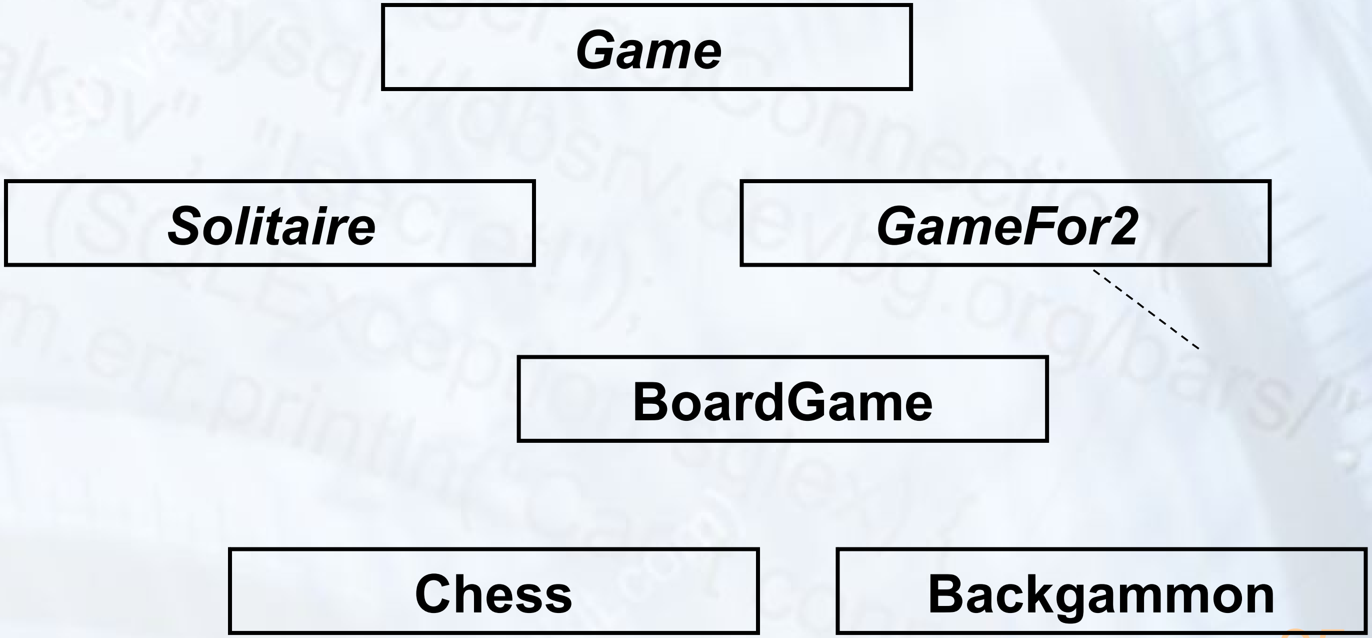
- An abstract class can have code for some of its methods
  - Other methods are declared abstract and left with no code
- An interface only lists methods but does not have any code
- A concrete class may extend an abstract class and/or implement one or several interfaces, supplying the code for all the methods

Inheritance Benefits

- Inheritance plays a dual role:
  - A subclass reuses the code from the superclass
  - A subclass inherits the *data type* of the superclass (or interface) as its own secondary type

Class Hierarchies

- **Inheritance leads to a hierarchy of classes and/or interfaces in an application:**



Game

Solitaire

GameFor2

BoardGame

Chess

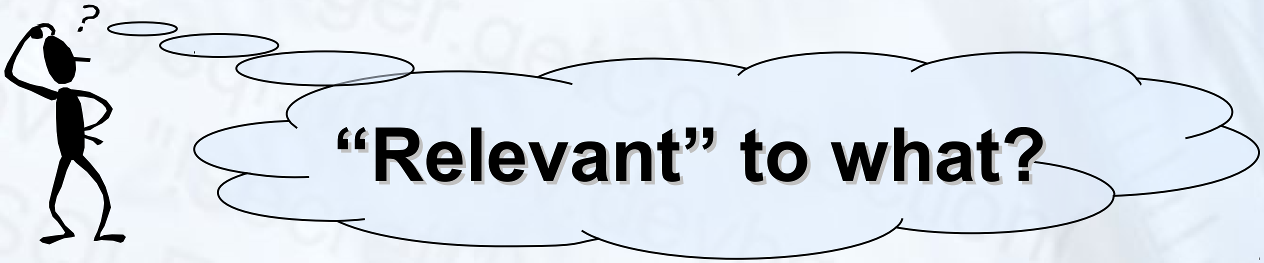
Backgammon

Inheritance

- An object of a class at the bottom of a hierarchy inherits all the methods of all the classes above
- It also inherits the data types of all the classes and interfaces above
- Inheritance is also used to extend hierarchies of library classes
  - Allows reusing the library code and inheriting library data types

Abstraction

- Abstraction means ignoring irrelevant features, properties, or functions and emphasizing the relevant ones...



- ... relevant to the given project (with an eye to future reuse in similar projects)
- Abstraction = managing complexity

Abstraction

- Abstraction is something we do every day
  - Looking at an object, we see those things about it that have meaning to us
  - We abstract the properties of the object, and keep only what we need
- Allows us to represent a complex reality in terms of a simplified model
- Abstraction highlights the properties of an entity that we are most interested in and hides the others

Abstraction in Java

- In Java abstraction is achieved by use of
  - Abstract classes
  - Interfaces

Abstract Data Types

- Abstract Data Types (ADT) are data types defined by a set of operations
- Examples:

Abstraction in AWT/Swing

- java.lang.Object
- |
- +--java.awt.Component
- |
- +--java.awt.Container
- |
- +--javax.swing.JComponent
- |
- +--javax.swing.AbstractButton

Encapsulation

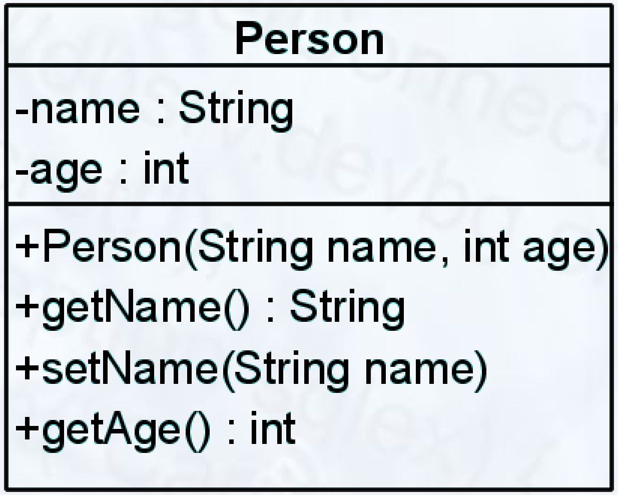
- Encapsulation means that all data members (*fields*) of a class are declared *private*
  - Some methods may be private, too
- The class interacts with other classes (called the *clients* of this class) only through the class's constructors and public methods
- Constructors and public methods of a class serve as the *interface* to class's clients

Encapsulation

- Ensures that structural changes remain *local*:
  - Usually, the internal structure of a class changes more often than the class's constructors and methods
  - Encapsulation ensures that when fields change, no changes are needed in other classes (a principle known as "locality")
- Hiding implementation details reduces complexity → easier maintenance

Encapsulation – Example

- Data Fields are private
- Constructors and accessor methods are defined



Person

-name : String
-age : int

+Person(String name, int age)
+getName() : String
+setName(String name)
+getAge() : int

Polymorphism

- Ability to take more than one form
  - A class can be used through its parent class's interface
  - A subclass may override the implementation of an operation it inherits from a superclass (late binding)
- Polymorphism allows abstract operations to be defined and used
  - Abstract operations are defined in the base class's interface and implemented in the subclasses

Polymorphism

- Why use an object as a more generic type?
  - To perform abstract operations
  - To mix different related types in the same collection
  - To pass it to a method that expects a parameter of a more generic type
  - To declare a more generic field (especially in an abstract class) which will be initialized and “specialized” later

Polymorphism – Example

Abstract class

Abstract action

Concrete class

Overriden action

Overriden action

```
Square::calcSurface() {
  return size * size;
}
```

```
Circle::calcSurface() {
  return PI * radius *
raduis;
}
```

Polymorphism

- **Polymorphism ensures that the appropriate method is called for an object of a specific type when the object is disguised as a more generic type:**

```
Figure f1 = new Square(...);
Figure f2 = new Circle(...);

// This will call Square::calcSurface()
int surface = f1.calcSurface();

// This will call Square::calcSurface()
int surface = f2.calcSurface();
```

Polymorphism in Java

- Good news: polymorphism is already supported in Java
  - All you have to do is use it properly
- Polymorphism is implemented using a technique called *late method binding*:
  - Exact method to call is determined at run time before performing the call

Questions?

Problems

1. Describe the term **object** in OOP.
2. Describe the term **class** in OOP.
3. Describe the term **interface** in OOP.
4. Describe the term **inheritance** in OOP.
5. Describe the term **abstraction** in OOP.
6. Describe the term **encapsulation** in OOP.
7. Describe the term **polymorphism** in OOP.