Basics of the Object Oriented Programming

Associate Professor Viorica Rozina Chifu
viorica.chifu@cs.utcluj.ro

About BOOP

- Lectures and Laboratory description will be available on Teams
- 14 lectures, 14 lab sessions
- Laboratory homework's, lab project, final exam

What is Object Oriented Programming?

- Object-orientation
  - A set of tools and methods used by software engineers to build reliable, reusable software systems that fulfills the requirements of its users
  - Provides a new view of computation
    - » A software system is seen as a community of objects that cooperate with each other by passing messages in solving a problem
- OOP language provides support for object-oriented concepts:
  - Objects and Classes
  - Encapsulation and Inheritance
  - Polymorphism and Dynamic binding

Programming Paradigms

- Object-oriented programming
  - Is one of several programming paradigms
- Other programming paradigms include:
  - Imperative programming paradigm
    - » e.g., Pascal or C languages
  - Logic programming paradigm
    - » e.g., Prolog language
  - Functional programming paradigm
    - » e.g., ML, Haskell or Lisp languages
- Note: Logic and functional languages are named declarative languages

Object Orientation as a New Paradigm

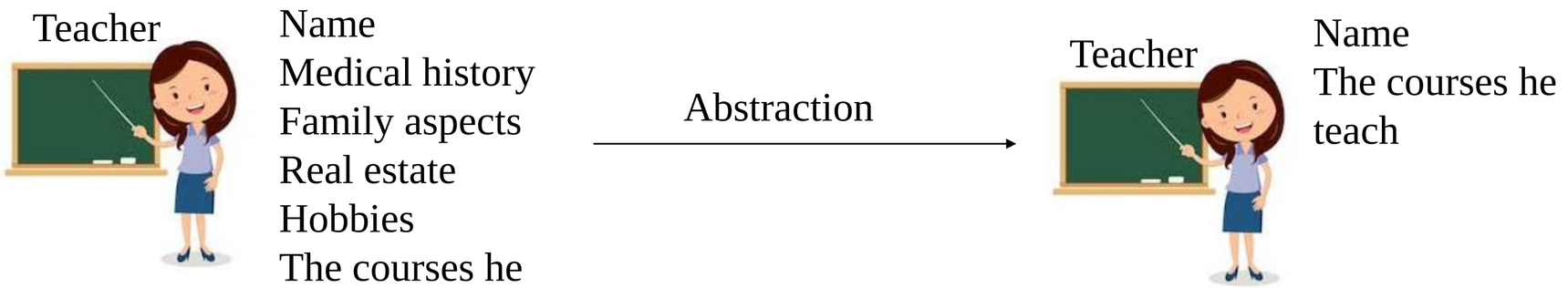
- The main concepts in OOP are:
  - Abstraction
  - Encapsulation
  - Inheritance
  - Polymorphism

Object Orientation as a New Paradigm - Abstraction

- Timothy Budd define **abstraction** as being the capacity of hiding some details of process in order to bring out more clearly other aspects, or structures
- Abstraction is the process of filtering out insignificant aspects about a thing/ person so that only the important aspects are left
- In the context of OOP, abstraction is expresses in:
  - **Objects** that are the fundamental logical blocks; each object is **instance** of a class
  - **Classes** that are a description of a collection of objects that have the same structure and similar behavior; classes can be related to inheritance
- Abstraction in OOP involves defining classes and objects in a way that hides complex implementation details and exposes only the relevant interfaces or methods for interacting with those objects.

Object Orientation as a New Paradigm - Abstraction

- Example of **abstraction**
  - When thinking about a TEACHER, we do not have in mind a particular teacher but an **abstract idea** of a teacher
    - A teacher has a *name*, a *medical history like any other person*, *can be married of not*, *can have children or not*, *have some hobbies*, *is the owner of some real estate*, and *teach some courses*; but which is important to distinguish a teacher from an ordinary person is it ***name*** and the ***courses*** he/she teach (this is **abstraction**)



Object Orientation as a New Paradigm - Encapsulation

- Encapsulation is one of the fundamental concepts in object-oriented programming (OOP)
- Encapsulation is the practice of grouping related data and behavior (methods) into a single unit (e.g., a class in Java) and then restricting access to that unit from the outside

Object Orientation as a New Paradigm - Encapsulation

- It refers to the process of **hiding the implementation** details of an object and exposing only the necessary information or functionality to the outside world.
- Encapsulation ensures that an object's internal state (i.e., its data) is not directly accessible from the outside and can only be modified through a set of predefined methods (i.e., its behavior).
- In any object-oriented programming language, **encapsulation** can be implemented by getter and setter methods and **private** access modifier used in attributes declaration
  - »As the names indicate, a *getter* method retrieves an attribute, and a *setter* method changes it
  - »Depending on the methods that you implement, you can decide if an attribute can be read and changed, or if it’s read-only, or if it is not visible at all

Object Orientation as a New Paradigm - Encapsulation

- **Encapsulation** is important in OOP because it helps to improve the maintainability, security, and usability of code.
- **Encapsulation** also helps to prevent unauthorized access to sensitive data and restricts the actions that can be performed on an object to a set of predefined methods, thus improving the security and reliability of the code.

Object Orientation as a New Paradigm - Encapsulation

- In conclusion, the benefits of encapsulation are:
  - **Promotes Information-hiding**:
    - »Encapsulation allows an object to hide its internal implementation details from other objects.
      - This means that an object's internal state can be protected and controlled, making the object more robust and reliable.
      - This allows the object to change its private implementation details at any time without impacting other objects that rely on its public interface
    - »Encapsulation allows the object's public interface to be designed independently of its implementation, which enhances the object's modularity and flexibility.
      - The objects have publicly available interfaces through which communicate with other objects
    - »By **hiding the implementation details** of an object, it makes it easier to change the implementation without affecting the code that uses the object

Object Orientation as a New Paradigm - Encapsulation

- In conclusion, the benefits of encapsulation are:
  - **Promotes Modularity**:
    - Encapsulation allows the source code for an object to be developed and maintained independently of the source code for other objects
      - This means that modifications to an object's implementation will not affect the functioning of other objects in the system.
      - Additionally, encapsulation allows objects to be easily passed around within the system, making them highly portable and flexible.

Object Orientation as a New Paradigm - Encapsulation

- **Modularity**
  - Modularity in object-oriented programming involves breaking down a program into separate, independent subunits or modules that can be compiled separately.
  - These modules are connected to each other, but the degree of connection is kept small so that any changes made to one module have minimal impact on the others.
  - However, the classes that belong to a module must be strongly connected to each other to justify grouping them together.
  - Programming languages that support the module concept distinguish between
    - » the interface of the module, which consists of elements such as types, variables, and functions that are visible in other modules,
    - » the implementation of the module, which contains elements that are only visible within the module itself.
  - This separation of interface and implementation provides greater flexibility in designing and maintaining software systems.
    - » By keeping modules separate and well-defined, code can be more easily reused, modified, and tested.

Object Orientation as a New Paradigm - Encapsulation

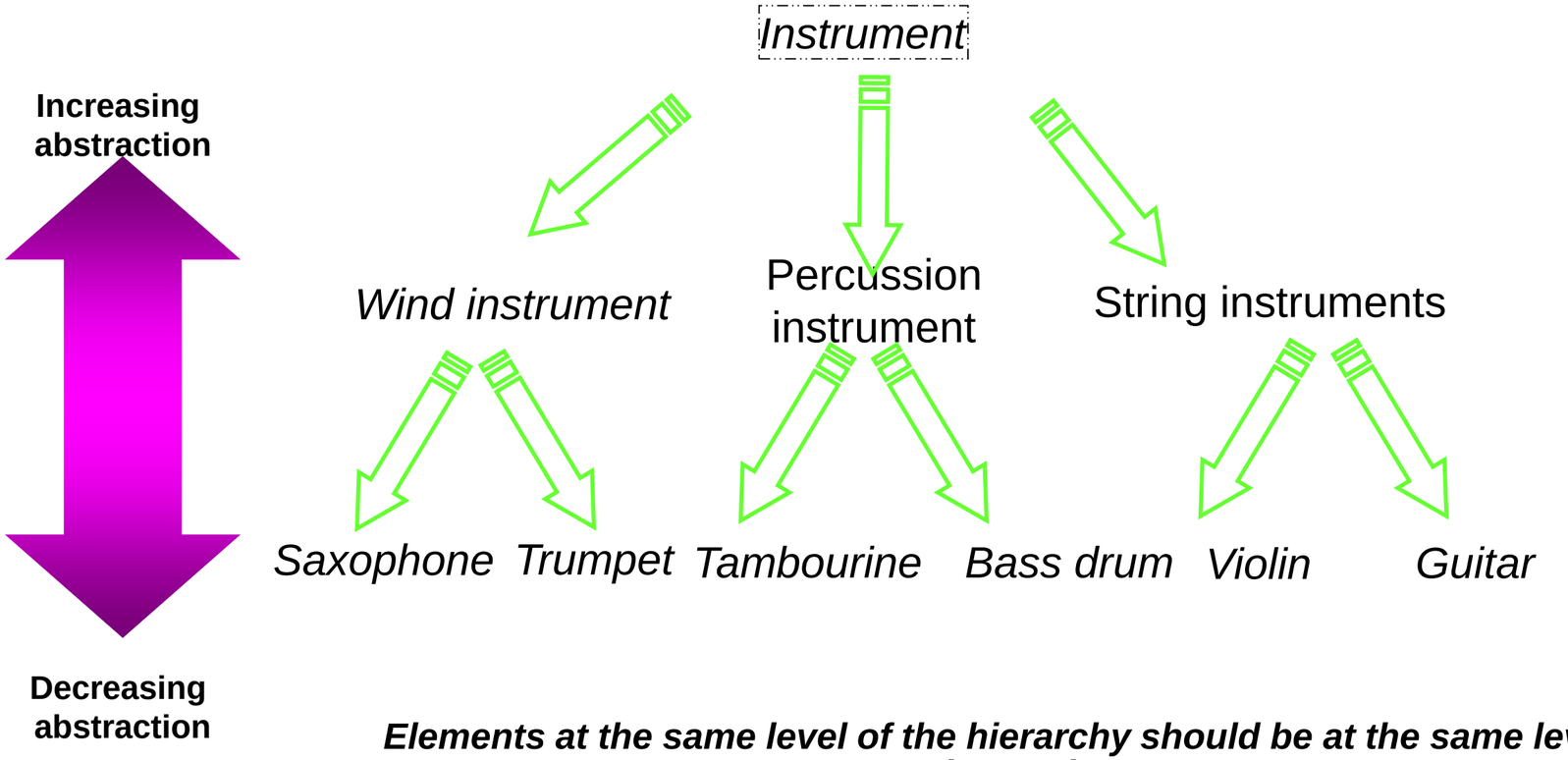
- Modularity (i.e., breaking down complex systems or programs into smaller, more manageable pieces) examples
  - A classic example of modularity is in building construction.
    - Instead of building a large structure all at once, builders divide the construction process into separate, modular components such as walls, floors, and ceilings.
    - These modules can be designed, constructed, and tested independently before being combined to form the final structure.
  - Another example of modularity is in software development, when developing a web application.
    - A web application might consist of separate modules for user authentication, database management, and front-end design.
    - These modules can be developed and tested independently and can be easily replaced or modified without affecting the other modules.
    - By breaking a complex system into smaller, well-defined modules, developers can more easily manage complexity, reduce errors, and improve overall system performance.

Object Orientation as a New Paradigm - **Inheritance**

- Inheritance refer to organizing classes into hierarchies, where a class might have one or more parent or child classes
- If a class has a parent class, we say it is derived or inherited from the parent class and it represents an “IS-A” type relationship
  - The child class “IS-A” type of the parent class
- Inheriting from another class allows a child class
  - to inherit a significant amount of functionality and properties from the parent class,
  - to add its own unique code and data.
- An important benefit of inheritance is that it promotes **code reuse**, as the child classes do not need to redefine the functions of the parent class.
  - This means that the child classes can leverage the functionality of the parent class and only add or modify what is needed to create their own specialized behavior.

Object Orientation as a New Paradigm - Hierarchy

- Example of Hierarchy of classes (i.e. of inheritance type)
  - Levels of abstraction



Elements at the same level of the hierarchy should be at the same level of abstraction

Object Orientation as a New Paradigm - Polymorphism

- Polymorphism is one of the core principles of object-oriented programming (OOP) that allows an object to take on multiple forms or behave differently in different contexts.
- In other words, polymorphism allows objects of different types to be treated as if they are of the same type
- In java, polymorphism is achieved through **inheritance** and **method overriding**.
  - When a child class inherits from a parent class, it can override the parent's methods to provide its own implementation.
  - This allows objects of the child class to be used interchangeably with objects of the parent class, as they share the same external interface.

Object Orientation as a New Paradigm - Polymorphism

- By allowing objects of different types to be treated as if they are of the same type, polymorphism enables programmers to write generic code that can work with a wide range of objects.
- Overall, polymorphism is a powerful and essential concept in OOP that helps to simplify the development of complex software systems by promoting code reuse, flexibility, and maintainability.

Object Orientation as a New Paradigm

- Object-oriented programming (OOP) utilizes problem-solving techniques that closely mimic the way humans solve day-to-day problems.
- OOP breaks down complex problems into smaller, more manageable components, and uses objects to represent those components and their interactions.

Object Orientation as a New Paradigm

- For example, consider the scenario where George wants to send flowers to his friend Anna, who lives in another city.
  - For this George calls to the nearest florist and give him the details about:
    - »The type of flowers to send and the address to which the flowers to be delivered
    - »George needs a confirmation that the flowers will be delivered

Object Orientation as a New Paradigm

- To solve this problem, *George* utilized a mechanism in which he found an agent, such as *Mary*, and conveyed his request through a message.
- *Mary* was responsible for fulfilling the request using various methods, but George did not need to know the specific methods used.
- George was not concerned with the details of how Mary fulfilled his request.
- However, upon investigation, he may have discovered that
  - *Mary* passed a slightly different message to another florist in the city where Anna lived.
  - That florist then passed on another message to a subordinate who arranged the flowers.
  - The flowers, along with yet another message, were then handed over to a delivery person, and so on.
  - Additionally, the florists had interactions with wholesalers, who in turn interacted with flower growers and so on.
- All these interactions were handled using various mechanisms and methods that were hidden from George's view.

Object Orientation as a New Paradigm

- This leads to the first conceptual picture of object-oriented programming:
  - An object-oriented program is structured as community of interacting objects (interacting objects)
    - Each object has a specific role to play within the program and performs actions that are utilized by other members of the community.
  - In an object-oriented program, each object has a distinct identity, encapsulating its own data and behavior.
  - Objects communicate with each other by exchanging messages, which contain information about the action that needs to be performed.
    - Each object in the program is responsible for a specific task, and collaborates with other objects to accomplish the overall goal of the program

Object Orientation as a New Paradigm

- In an object-oriented environment, community members interact by sending requests to each other.
- To start a particular action, a message is sent to the object that is responsible for that action.
- The message encodes the request and includes any additional information, such as arguments or parameters, that are necessary to execute the request.
- The object that receives the message is known as the receiver.
- If the receiver accepts the message, it takes on the responsibility of carrying out the requested action. I
- In response to the message, the receiver performs a method that satisfies the request.

Object Orientation as a New Paradigm

- The client sending the request does not have to be aware of how the request is executed.
- The client only needs to know the details of the request and what information needs to be supplied.
- The interpretation of the message is dependent on the receiver and can differ across different receivers.
- Distinct objects can interpret the same message in a different way based on their distinct responsibilities and behaviors.
- This flexibility allows for the creation of complex, modular systems that can adapt to changing requirements and scenarios.

Object Orientation as a New Paradigm - Messages and Responsibilities

- Object-oriented programming describes behavior in terms of responsibilities.
  - Requests made by clients for actions only specify the desired outcomes.
  - Receivers have the flexibility to use any technique that accomplishes the desired results.
  - This way of thinking enables greater independence between objects.

Object Orientation as a New Paradigm

- Advantages of OOPs (Object-Oriented Programming System):
  - *Easy to Understand and Modular Structure*:
    - » OOPs concepts provide a clear and easy-to-understand structure for programs.
    - »Objects are created to represent real-world entities, and each object has a specific role or responsibility.
    - »This makes it easy to break down complex problems into smaller, more manageable pieces, making the code more modular and easier to understand.
  - *Reusability*:
    - » Objects created for one OOP program can be reused in other programs, saving significant development time and cost.
    - »This is because objects can be designed to be independent and self-contained, making them easy to integrate into new programs.

Object Orientation as a New Paradigm

- Advantages of OOPs (Object-Oriented Programming System):
  - *Reduced Errors*:
    - » Large programs can be difficult to write and debug, but OOPs can help reduce errors by providing a clear and modular structure for code.
    - » Each object is responsible for a specific task, and this reduces the likelihood of errors and simplifies debugging.
  - *Program Modularity*:
    - » OOPs enhances program modularity because each object exists independently and can be reused in other programs.
    - » This makes it easy to make changes to one part of the code without affecting the rest of the program.
  - *Code Maintenance*:
    - » OOPs makes it easy to maintain code over time because objects can be modified or replaced without affecting the rest of the program.
    - » This helps to reduce maintenance costs and makes it easier to update code as needed.

Object Orientation as a New Paradigm - Classes and instances

- The main building blocks of OOP are classes and objects
  - Classes are used as templates to create **objects**, also known as **instances**.
  - Each **instance** has its own set of values for the properties defined in the class and can perform **actions** defined in the **class methods**.
  - The **class** defines the **behavior** of the **objects** instantiated from it, including which methods can be called on them and how they respond to those methods.
  - Therefore, all objects of a given class share the same methods and properties defined in the class, and invoking a method on an object triggers the execution of the corresponding method in the class.

Object Orientation as a New Paradigm - Classes and instances

- Object
  - Is modeled after real-world objects
    - »Have **state** and **behavior**
      - For example, **Students** objects have
        - **State** (*name*, *student number*, *courses* they are registered for)
        - **Behavior** (*take tests*, *attend courses*, *write tests*)
    - »Have an identity
  - Maintains its **state** in one or more **variables**
    - »A variable is an item of data having associated an identifier
  - Implements its **behavior** with **methods**
    - »A method is a function associated with an object

Fundamentals of Objects and Classes - How objects differ from classes?

- A class is a description of a **group** of objects with common properties (attributes), behavior (operations)
- A class is an ‘abstraction’ of objects (Is used to create object)
- A class is an **abstraction** in that it:
  - Emphasizes relevant characteristics
  - Suppresses other characteristics
  - It defines the **structure and behavior** of each object in the class
    - »We can tell a lot about how Mary will behave by understanding how Florists behave. We know, for example, that Mary, like all florists can arrange and deliver flowers
  - It serves as a **template** for creating objects
- Defines the **variables** (fields) and the **methods** common to all objects of a certain kind
- Declares instance variables that contain the **state** of every object
- Provide implementations for the instance methods necessary to operate on the **state** of the object

Fundamentals of Objects and Classes - How objects differ from classes?

- Objects are specific instances of a class that exist in memory at runtime and have their own unique state.
- Objects are created and destroyed as the program runs
- There can be many objects with the same structure, if they are created using the same class

Class Members and Instance Members

- Variables/methods of a class can be declared as **static** or **non-static**

static double *lastStudentNumber*; //static member/variable/field
double *studentNumber*; //a non−static variable
static void *printLastNumber*() {...} //static member/method
void *printNumber*() {...} //a non−static method

Class Members and Instance Members

- Instance variables and methods
  - Are the non-static members of a class in object-oriented programming.
  - They hold the state for a specific object.
- Since there may be many instances of a class, each instance has its own instance variables and distinct values stored in them.
- When an instance of a class is created, the system allocates memory for the object and all its instance variables.
- For example, in a class called **Students**, there may be multiple **Student** objects, each with a different number stored in its **Student Number** variable.
- Instance methods must be called on a specific instance of a class

Class Members and Instance Members

- Class variables and methods are static members of a class in object-oriented programming.
- When a member of a class is declared as static, it means that there is only one copy of that member, regardless of how many objects of the class are created.
- Class variables contain information that is shared by all objects of the class, while instance variables contain state specific to a particular object.
- If one object changes a class variable, it changes for all other objects of that type.
- Similarly, class methods are methods that are associated with the class itself, rather than with a specific instance of the class.
- They can be invoked directly from the class, without the need to create an instance of the class.
- An example of a class with class variables and methods is the Math class in Java, which contains static methods such as Math.sqrt(x) for computing the square root of a number.

Class Members and Instance Members - Example of static member

```
class UserData
  {
    static String name;
    static int age;
   }
```

- Since the **name** and **age** variables are declared as static, there will be only one copy of each variable that is shared by all instances of the **UserData** class.
- These variables will exist as long as the program runs and can be accessed directly through the class name (UserData.name and UserData.age).

Class Members and Instance Members - Example of instance member

```
class PlayerData
  {
     String name;
     int age;
   }
```

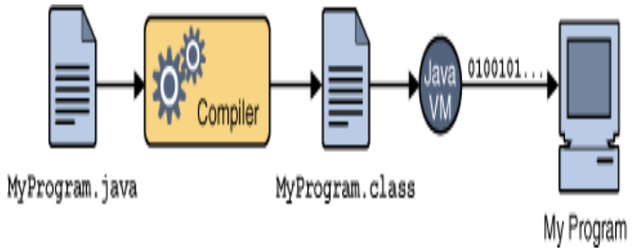
- In the **PlayerData** class, **name** and **age** are instance variables because they are not declared as static.
- This means that each object created from the class will have its own copy of the **name** and **age** variables.
- So, if you create two **PlayerData** objects, each object will have its own **name** and **age** values that can be different from the other object's values.

Class Members and Instance Members

- It is possible to have both static and non-static members in a single class.
  - Static members specify things that are associated with the class itself, rather than with any particular instance of the class.
    - » For example
      - A static variable might keep track of the total number of instances of the class that have been created.
      - A static method might perform a calculation that doesn't depend on the state of any particular instance of the class.
  - Non-static members specify things that are specific to each instance of the class.
    - »For example
      - A non-static variable might store the name of a person, and each instance of the class would have its own name.
      - A non-static method might perform an action that depends on the state of a particular instance of the class, such as updating the person's age.

About Java

- The Java programming language uses a compilation process where the source code is first written in plain text files with a .java extension
- The source code is then compiled using the **javac** compiler, which generates bytecode in the form of .class files.
  - Bytecode is a low-level, platform-independent set of instructions that can be executed by the Java Virtual Machine (JVM).
- When a Java program is executed, the JVM interprets the bytecode instructions and converts them into machine code that can be run by the host computer's processor.
- This approach allows Java programs to be written once and run on any platform that has a JVM installed.

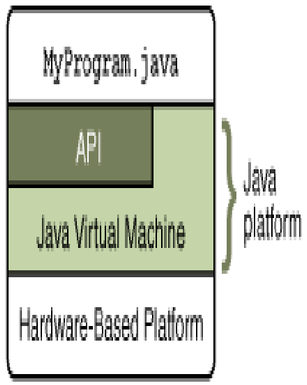


Java Platform

- Platform
  - Is the hardware or software environment in which a program runs
  - Some of the most popular platforms are:
    - Microsoft Windows
    - Linux
    - Solaris OS
    - Mac OS
  - Most platforms can be described as a combination of the operating system and underlying hardware
- Java platform
  - Is a software-only platform that runs on top of other hardware-based platforms
  - Has two components:
    - *Java Virtual Machine*
    - *Java Application Programming Interface* (API)

Java Platform

- Java Virtual Machine
  - Is the base for the Java platform
  - Is ported onto various hardware-based platforms
- API
  - Is a collection of software components that provide many useful capabilities
  - Is grouped into libraries of related classes and interfaces
    - These libraries are known as *packages*
- The API and Java Virtual Machine insulate the program from the underlying hardware



Java Platform

- Main benefices of java platform
  - Simplicity – java eliminates operators overloading, multiple inheritance and all the facilities that can make the code writing confusing
  - Robustness – java eliminates common sources of errors that occur because of pointers (pointers was eliminated in java), java assures automatic memory management, java eliminates cracks memory through by introducing a garbage collector
  - Complete object oriented – java completely eliminates procedural programming style
  - Easy to used in network programming
  - Security – java provides strict security mechanisms embodied in the programs
  - Portability – java is an independent language of work platform (the same application can run on different operating systems such as Windows, UNIX or Macintosh without any modification)
  - Allows multithreading programming
  - Is modelled like C and C++, so that the transition from C, C ++ Java is made very easy

Java Application Programs

- Types of Java programs
  - Stand-alone applications
    - Java application program that contains a class having a method named ***main***
    - When a Java application program is run, the run-time system automatically invokes the ***main*** method
    - All Java application programs start with the ***main*** method
    - May use a windowing interface or console (i.e., text) I/O
  - Applets
    - A Java program that run in a Web browser
    - Can be run from a location on the Internet
    - Can also be run with an applet viewer program for debugging
    - Applets always use a windowing interface

Standard Java Naming Conventions

- Packages
  - Names should be in lowercase
  - Gives simple names for the packages in small projects:

    package *pokeranalyzer*

    package *mycalculator*
  - Subdivides the names in the case of large projects because the packages might be imported into other classes

    »Package's name should start with the company domain before being split into layers or features:

    package *com.mycompany.utilities*

Standard Java Naming Conventions

- Classes
  - Names should be in upper camel case
  - Use nouns because a class represents something in the real world:

    class *Customer*

    class *Account*
- Interfaces
  - Names should be in camel case
  - Names describes an operation that a class implementing the interface can do:

    interface *Comparable*

    interface *Enumerable*
  - Some programmers like to distinguish interfaces by beginning the name with an “I“

    interface *IComparable*

    interface *IEnumerable*

Standard Java Naming Conventions

- Methods
  - Names should be in mixed case
  - Use verbs to describe what the method does:

    void *calculateTax*()

    String *getSurname*()
- Variables
  - Names should be in mixed case
  - Names should specify what the value of the variable represents

    String *firstName*

    int *orderNumber*
  - Use very short names when the variables are short lived, such as in **for** loops:

```
for (int i=0; i<20;i++) {//i only lives in here}
```

Standard Java Naming Conventions

- Constants
  - Names should be in uppercase

    static final int DEFAULT_WIDTH

    static final int MAX_HEIGHT

Java Types

- JAVA classifies values and expressions into types
  - e.g., **String** and **int** are types
- A type specifies the allowed values and allowed operations on values of that type

Java Types

- Advantages of using types by a programming language
  - **Safety**
    - » Use of types may allow a compiler to detect meaningless or invalid code
      - e.g., an expression of type ”Hello, World” / 3 is invalid because one cannot divide a string literal by an integer
  - **Optimization**
    - » Static type-checking may provide useful information to a compiler
      - Compiler may then be able to generate more efficient code
  - **Documentation**
    - » Types can serve as a form of documentation, since they can illustrate the intent of the programmer
  - **Abstraction**
    - » Types allow programmers to think about programs at a higher level
      - e.g., programmers can think of strings as values instead of as an array of bytes

Java Types

- There are two types in JAVA
  - Primitive types
    - » Boolean, char, short, int, long, float, double
  - Reference types
- Any variable can be declared either as a primitive type or as a reference type