**Defining Classes**

* **Abstract Data Type :**
  + Data type whose representation is hidden from the client

String ADT – indexed sequence of chars:

String()

int length()

char charAt(int index)

boolean isEmpty()

*// many others…*

* + You **don`t** need to know the **implementation** to use an abstractive data type.
* Defining simple classes
  + Class is a concrete implementation of an abstractive data type
  + Classes provide structure for describing and creating objects
* Naming classes
  + Classes should be PascalCase
  + Use descriptive nouns
  + Avoid abbreviations ( except widely known, e.g. URL, HTTP, etc.)

class Dice { … }

class BankAccount { … }

class IntegerCalculator { … }

* Class Members
  + Class is made up of state and behavior
  + Fields store state
  + Methods describe behavior

class Dice {

int sides; Fields

String type;

void roll(){ … }

} Method

* Creating an Object
  + A class can have many instances (objects)

class Program {

public static void main(String args) {

Dice diceD6 = new Dice();

Dice diceD8 = new Dice();

}

} Use the new

keyoword

Variable stores

a reference

* Object reference
  + Declaring a variable creates a **reference** in the stack
  + **new** keyword allocates memory on the heap

diceD6 Reference has a fixed size

(1540e19d)

**Stack**

**Dice diceD6 = new Dice();**

Type = null

Sides = 0 State is kept in the heap

**Heap**

* Fields
  + Class fields have type and name

**class Dice {**

**String type;**

**int sides;**

**int[] rollFrequency;**

**Person owner;**

**…**

**}**

* UML Diagram
  + ‘+’ – means that next variable will be public
  + ‘-‘ – means that next variable will be private
  + Underlined fields – means that variable must be static

**BankAccount**

**+id:int - > Fields**

**+balance:double**

**(no actions) - > Methods**

* Modifiers
  + Classes and class members have modifiers
  + Modifiers define visibility

public class Dice {

private int sides;

public void roll(int amount);

}

* + - Fields should always be private!
* Methods
  + Store executable code (algorithm) that manipulate state

class Dice {

private int sides;

public int roll() { **this** points to the current instance

Random rnd = new Random();

int rollResult = nextInt(**this**.sides) + 1;   
 return rollResult;

}

}

* Getters and Setters
  + Used to create accessors and mutators (getters and setters)
    - Field is hidden
    - Getter provides access to field
    - Setter provide field change ( Validations must be here )

class Dice {

**private** int sides; **Field**

**public** int getSides() {

return this.sides; **Getter**

}

**public** void setSides(int sides) {  
 **this**.sides = sides; **Setter**

}

}

* Constructors
  + Special methods, executed during object creation.

class Dice {

int sides;

public Dice() {

this.sides = 6; **Overloading default constructor**

**} Constructor without parameter**

}

* + You can have multiple constructors in the same class

public Dice(int sides) {

this.sides = sides; **Constructor with parameters**

}

* Constructor Chaining
  + Constructors can call each other

class Dice {

Calls constructor with parameters

int sides;

public Dice() {

**this(6);**

( 6 ) should be declared in a final variable

}

public **Dice**(int sides) {

this.sides = sides; **this();** - calls the default

} constructor

}

* + Constructors set object`s initial state
* Static Members
  + Static members are shared class-wide. We could have a static integer, which counts the objects of our class. Instances doesn`t have information about this static integer.

class BankAccount {

private static int accountsCount; example 1;

public BankAccount() {

accountsCount++;

} .

…

}

class BankAccount {

private static double interestRate; example 2;

public static setInterestRate(doube rate) {

interestRate = rate;

}

…

}

* Accessing Static Members
  + Access static members through the class name
  + You don`t need an instance

class Program {

public static void main(String[] args) {

BankAccount.setInterestRate(2.2);

}

Sets the rate for all bank accounts

}