University American College Skopje

School of Computer Science and Information Technologies

Course: Programming Languages

2021/2022

Classes. Arrays of Reference Type Objects. Command Prompt Input

Lecture 3

TYPES OF LANGUAGES

- There are many ways in which programming languages can be differentiated among each other.
- By their levels of abstraction:
  - Machine languages
  - Assembly languages
  - High level languages
- By their execution types:
  - Compiled
  - Interpreted
- *By their orientations:*
  - *Procedural*
  - *Object oriented*

LANGUAGE ORIENTATIONS

Function oriented languages (procedural languages):

- Programming is oriented towards actions
- The unit of programming is the function
  - Group of actions = function
  - Group of functions = program
- FORTRAN, BASIC, Pascal, C…

Language Orientations

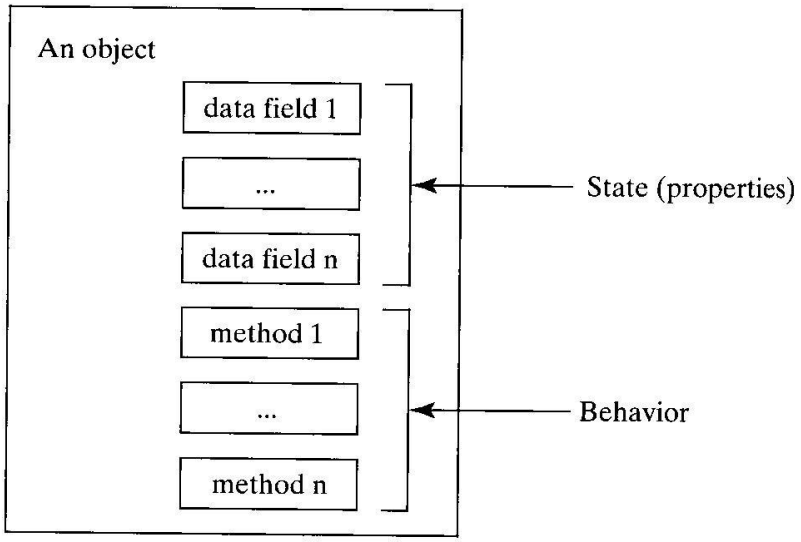
Object oriented languages:

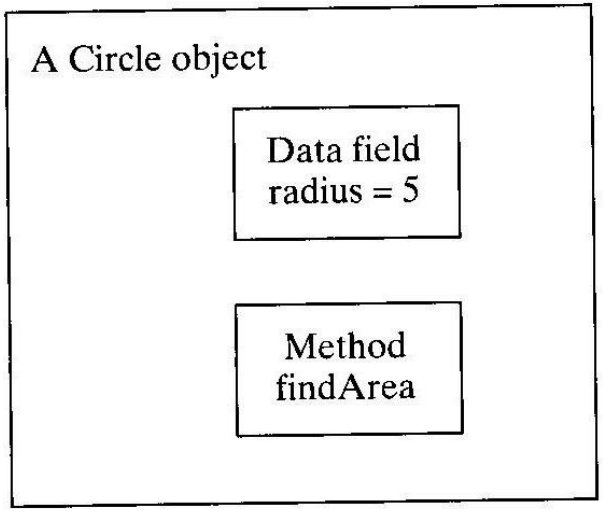
- Programming is oriented towards entities
- The unit of programming is the class, from which objects are instantiated, which perform functions
- Program = group of classes which contain data and methods, which are instantiated into objects with attributes and behaviors and thus perform the actions
- C++, Java, C#, VisualBasic, Delphi…

OBJECTS

- Object is an entity in the real world which can be unambiguously identified.
  - Student, table, car, loan,…
- The object has:
  - An unique identity – an identifier by which it is differentiated from the other parts of the code;
  - A state – a collection of fields and their values;
  - Behaviors – the methods of the object.

OBJECTS


An object
data field 1
...
data field n
method 1
...
method n
State (properties)
Behavior


A Circle object
Data field
radius = 5
Method
findArea

CLASSES

- The **classes are blueprints for the objects**
- The **classes define new types of data**
- These new types of data contain fields (attributes) and methods (behaviors).
- Many objects can be generated from a single class.
  - E.g. if you have a recipe for an apple pie (class), you can create many apple pies (objects).
- **Creating an object from a class is called *instantiation***
  - The term "object" and "instance (of a class)" are often used as synonyms.

CLASSES

```
class  nameOfClass
{
    //fields declarations
    type variable1;
    type variable2;
    ….
    //methods declarations
    returnType method1(arguments)
    {
        //method1 body
    }

    returnType method2(arguments)
    {
        //method2 body
    }
    ….
}
```

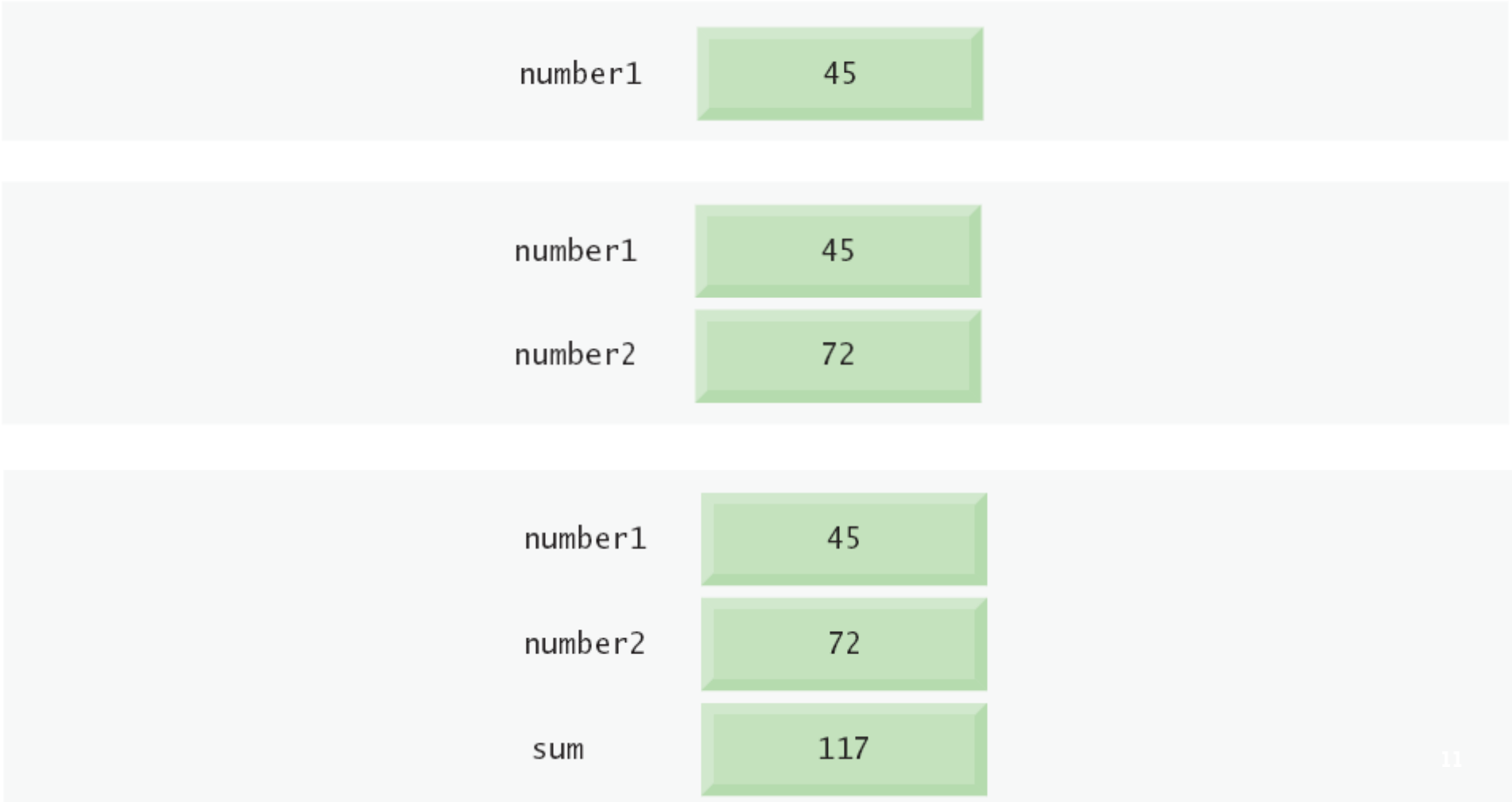
CLASSES

Example:

```
class Circle
{
  //fields declarations
  int radius;          // circle radius
  String color;        // circle color
  …

  //methods declarations
  Circle()             // constructor
  {…}
  double getArea()     // circle area
  {…}
  …
}
```

MEMORY CONCEPTS

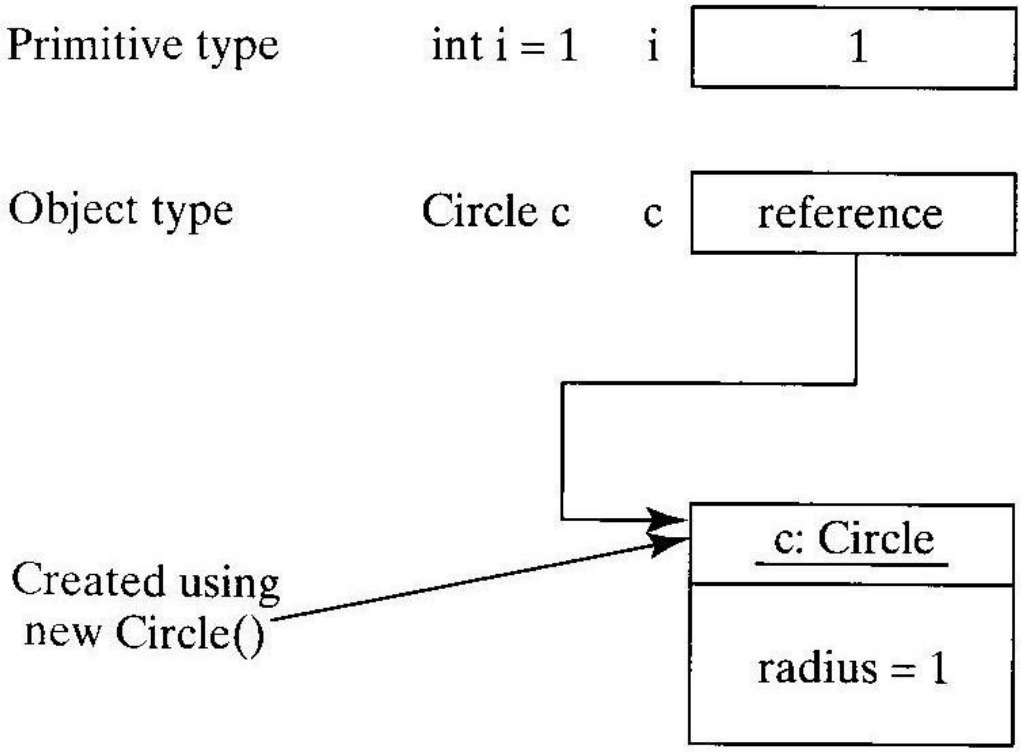
- The names of the variables coincide with locations in the computer memory
- Every variable has a name, type and value
- When the value is set to a memory location, the value replaces the previously set value on that location
- When the value is read from a memory location, the process is nondestructive


number1
45
number1
45
number2
72
number1
45
number2
72
sum
117

MEMORY MANAGEMENT

- In Java, memory locations are managed differently for object of primitive type and of reference type
- For a primitive type object, the memory address that is accessed using an identifier contains the value of the primitive type
- For a reference type object, the memory address that is accessed using an identifier contains a reference to where the object is located
  - Every object that is instantiated by a class is a reference type object.
  - References in Java are (almost) like pointers in C++.

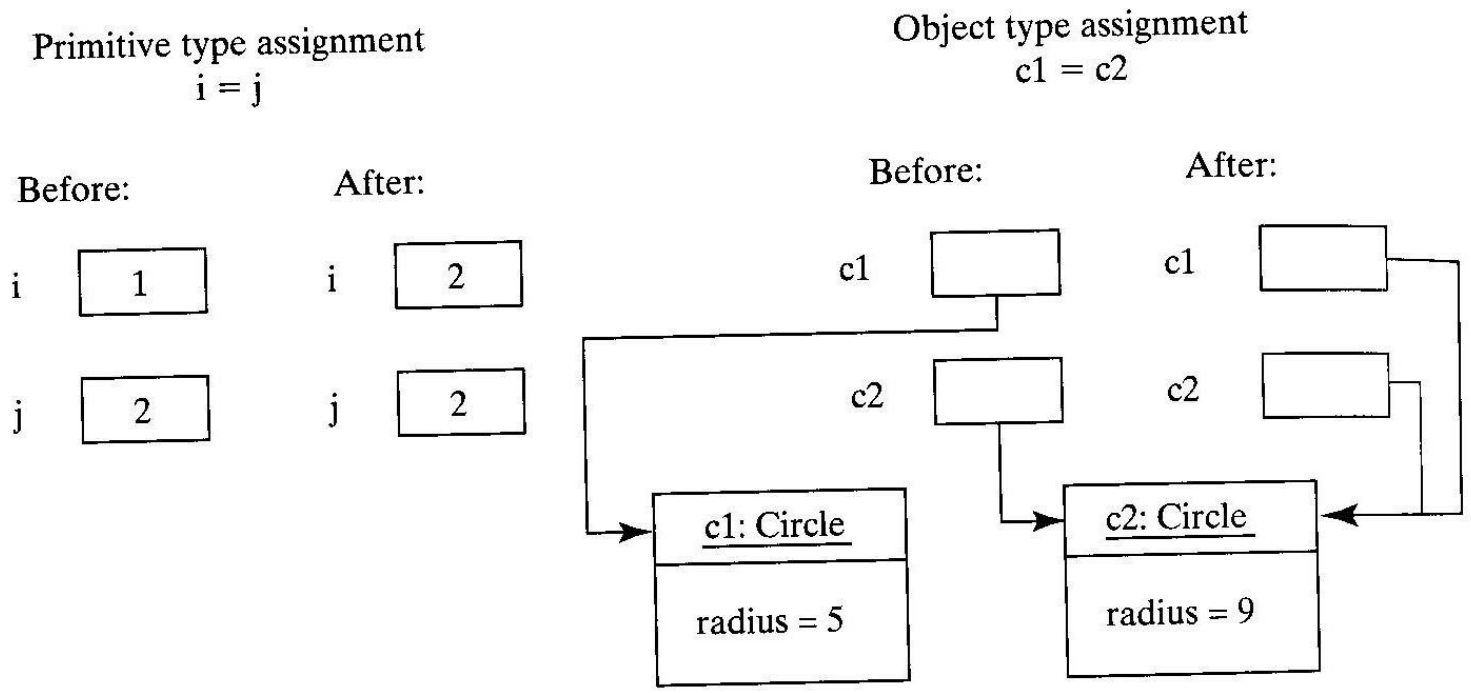
MEMORY MANAGEMENT



MEMORY MANAGEMENT

- Also, when objects of **two primitive** types are **equated**, the value of one of the objects is copied into the other
- When the **objects** of two reference types are **equated**, the reference from one of the objects is copied onto the other reference, which by default doesn't affect the object that is actually stored in the memory.
  - Access using references = *indirect access*

MEMORY MANAGEMENT



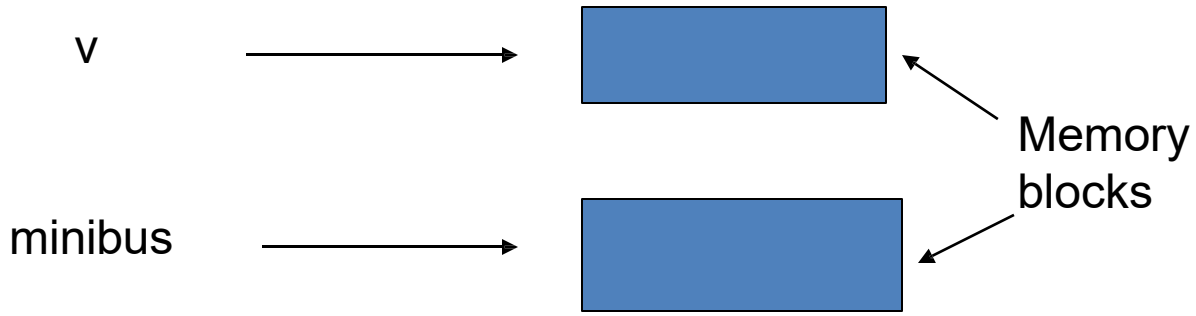
MEMORY MANAGEMENT

- If a reference that previously referenced an object is set to reference a different object (or nothing), the preceding object **becomes** ***garbage***
- The runtime system detects the appearance of such *garbage* and it automatically reclaims it as reusable memory.
  - This process is called *garbage collection.*
  - In C++ it takes destructors to release the memory; in Java, this process is automated.

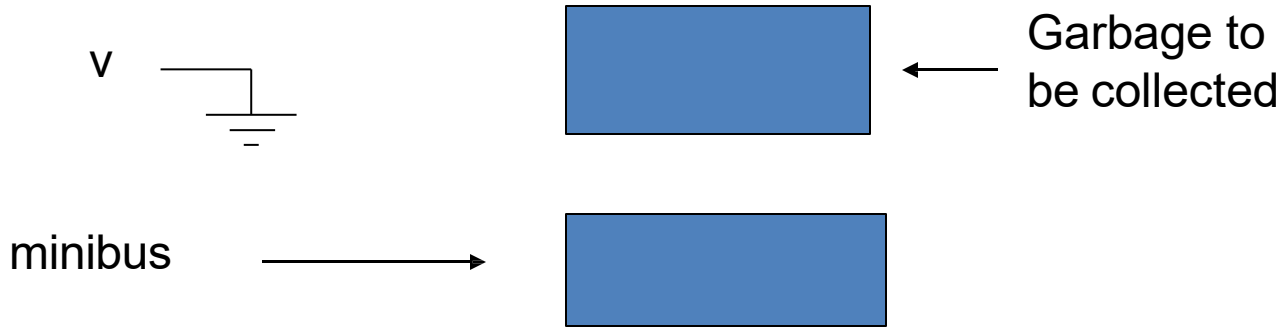
THE NULL VALUE

- When a reference to an object is declared, its default value is null.
- The nullvalue doesn't need to be cast to a certain type – it is accepted by all referent types as a valid value.
- Its meaning is that the reference points to nothing, i.e., that it doesn't point to an object of its own type.
- If the reference used to point to an object, assigning a null value is a call to the garbage collector to reclaim the memory previously occupied by that object.

```
minibus = new Vehicle();
v = new Vehicle();
```



```
v = null;
```



METHODS

- Methods are segments of code which process the object data
- One method may invoke another method from the same class or from a different class
  - The objects communicate among themselves through method invocations
- The instance methods (or regular methods) are created and embedded into each created object
- The class methods (or static methods) affect all methods of that class at the same time.
  - Therefore, there is only one copy of that method per class, which is the reason why that method is invoked through the class, instead of an instance of that class (i.e., object).

General form of a method

```
returnType methodName(argumentList)
{
   // method body
   return valueToBeReturned;
}
```

- *`methodName`* is any valid Java identifier.
- *`returnType`* is any primitive or reference type, or void.
- *`argumentList`* contains the arguments of the method, listed by their types and formal names; if there is more than one argument, the arguments are delimited using commas (,) .
  - E.g. `void someMethod(int p1, double p2, boolean p3)` – 3 parameters
- *`valueToBeReturned`* is a value of type *returnType*
  - If *returnType* is void, *valueToBeReturned* or even the entire returnexpression can be omitted.

METHOD INVOCATION

- Generally, the method is invoked by its qualification with a reference of an object containing that method.
- *The method doesn't exist on its own* – it must be a part of a class.
- Example:

getArea()is a method of the Circle class so object ccan invoke it.

```
c.getArea();
```

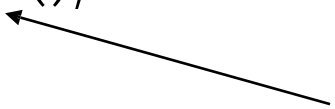
The methods are qualified by adding a period (.) between the object which invokes that methods and the name of the method (along with the method parameters).

METHOD INVOCATION

- If the method is defined as static, it is invoked by qualifying it with the name of the class, instead of an object of that class.

- Example:

Math.random();



A staticmethod – it is accessed through the class itself, instead of an object of that class.

THE MAIN() METHOD

- Method overview:

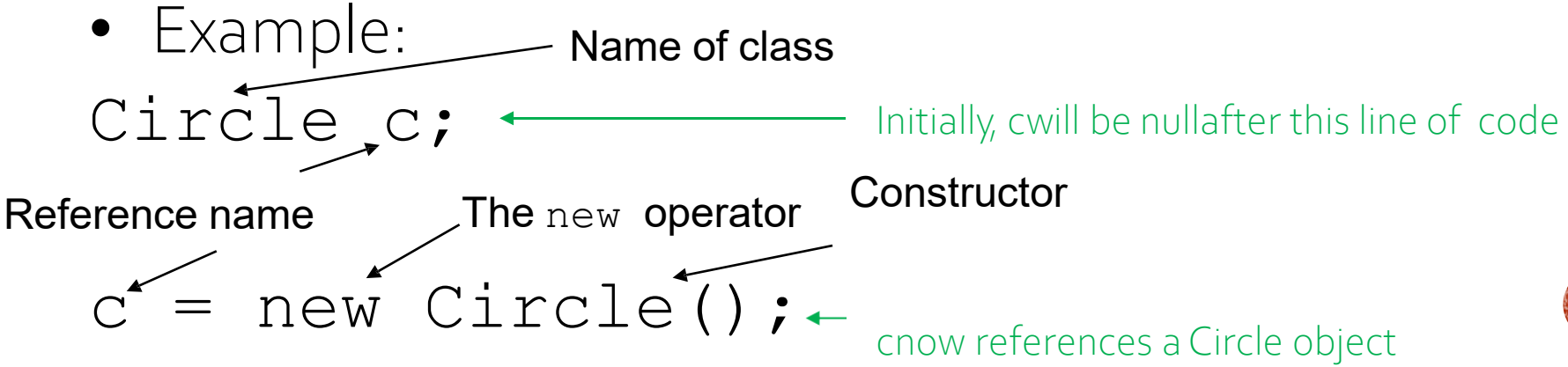
public static void main(String[] args)

  - Can be accessed from outside the class
  - A class method – there is only one copy of it in the memory
  - Returns no value
  - Accepts only one String[] parameter

THE NEW OPERATOR

- To create an object, i.e., a class instance, the new operator is used
- This operator claims sufficient memory so that an instance of any class (i.e., object) would be stored.
- That object is then referenced through a reference of that object.

- Example:



CONSTRUCTOR

- The *constructor* is a special method of the class.
- It is invoked immediately upon the object reference.
- Its primary function is to initialize the data fields of the object to specific values.
- *The constructor does not create the object* – it only prepares it for functioning.

General form of a constructor

```
className(argumentList)
{
   // method body
}
```

- *className* is the name of the class (the name of the constructor is identical to the name of the class).
- *argumentList* contains the method arguments, listed by their types and formal names; if there are more than one argument, they are delimited using commas (,) .
- The constructor returns nothing, and is not even void(since it only initializes the object fields and is invoked only once, upon object generation using the newoperator – it cannot be invoked by request).

CONSTRUCTORS

- The class can have multiple constructors.
- The constructor without parameters is called the *default constructor.*
- If no constructor is defined in the class, the default constructor will be implemented by the compiler and it will not contain its own code.
  - Thus, the class will be able to be instantiated.

CONSTRUCTORS

- Constructors, like other methods, may accept parameters, which will then be used in the method body.
- *Constructor overloading* is a process of creating several constructors in the class, which would be used in various occasions.
  - It is a special case of *method overloading*, which is a subsequent topic.

CONSTRUCTORS

- If the parameterized constructor (i.e., the constructor containing parameters) is defined, the default (parameterless) constructor will *not* be defined.
- Java considers that the programmer *knows what s/he is doing*.
  - If the class doesn't contain a default constructor, but contains another, it is because the programmer *doesn't need* the default constructor.

ARRAYS OF REFERENCE TYPE OBJECTS

- An array can be created for any object type, primitive or referent.
- The arrays themselves are objects of type *objectType[].*
  - i.e., an array of *objectType* elements is an object of type *objectType[].*
  - *objectType* can be either a primitive type or a referent type value.
    - It cannot be null or void.

ARRAYS OF REFERENCE TYPE OBJECTS

- Example:

```
Circle[] cArr = new Circle[numberOfElements];
```

- This means that:
  - *numberOfElements* must be an integer type value (byte, short, int, long) and have a greater value than zero.
    - The array cannot have e.g. -53.67 or true elements.
  - cArr will be an object of type Circle[]
  - cArr will contain objects of type Circle

ARRAYS OF REFERENCE TYPE OBJECTS

- Example:

```
Circle[] cArr = new Circle[numberOfElements];
```

- This means that (continued) :
  - Initially, all elements of the array (cArr[0], cArr[1], … cArr[numberOfElements – 1]) will have the null value, until they get initialized to other values.

    - **E.g.,**
```
cArr[0] = new Circle();
cArr[1] = new Circle(someParameters);
cArr[2] = cArr[1];
```
      etc.

ARRAYS OF REFERENCE TYPE OBJECTS

- Once the elements of the array are initialized, their fields and methods can be accessed as all other elements.

- Example:

```
System.out.println("The are of the circle" +
  " with index 2 is " + cArr[2].getArea());
```


Invoking a method from an array element

ARRAYS OF REFERENCE TYPE OBJECTS

- The length attribute is valid for arrays of all types of values, primitive or referent.
- Direct definition of arrays of reference type objects is also permitted.
  - Example:

```
Circle[] cArr2 =
{new Circle(), new Circle(parameters)};
```

This array will contain 2 elements which will be immediately created.

- Arrays of reference type objects can be multidimensional and jagged as well.

COMMAND PROMPT INPUT

- The command prompt (CMD) can be used to input arguments upon the start of the JVM.
  - JVM is started using the program java.exe which is the Java interpreter
- The definition of the main()method is:

```
public static void main(String[] args)
{…}
```

  - It has a String[] argument, which is an array of parameters on the command line.
  - If arguments are not passed to the program upon its start, this array is null.

COMMAND PROMPT INPUT

- Example:

```
java Program   2      "SCSI student"      UACS
```

arg[0] arg[1] arg[2]

- An arbitrary number of parameters can be input upon the program start.
  - The String[] args parameter will be created directly upon program start.
  - Thus, the parameters will be accessible as elements of this array inside the main() method.

COMMAND PROMPT INPUT

- Important:

Parameters may be passed inside the program only as String objects.

  - If an object of a different type is passed into the program, it is automatically cast to a String.
- To use the arguments in the form in which they were intended to be used in the program, the appropriate Stringarguments must be cast into the preferred types.
- Also, only parts of the String arguments may be necessary – they are obtained through *string parsing.*

USE CASE – SHOPPING CART

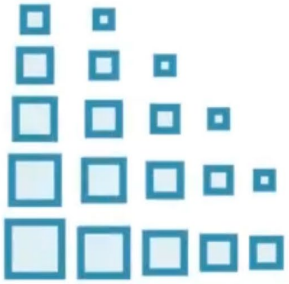
FROM USE CASES TO CONCEPTUAL CLASSES

- Scenarios of system operation
- Often started by a human user
- List the sequence of interaction steps

CREATE ORDER AND CHECK OUT

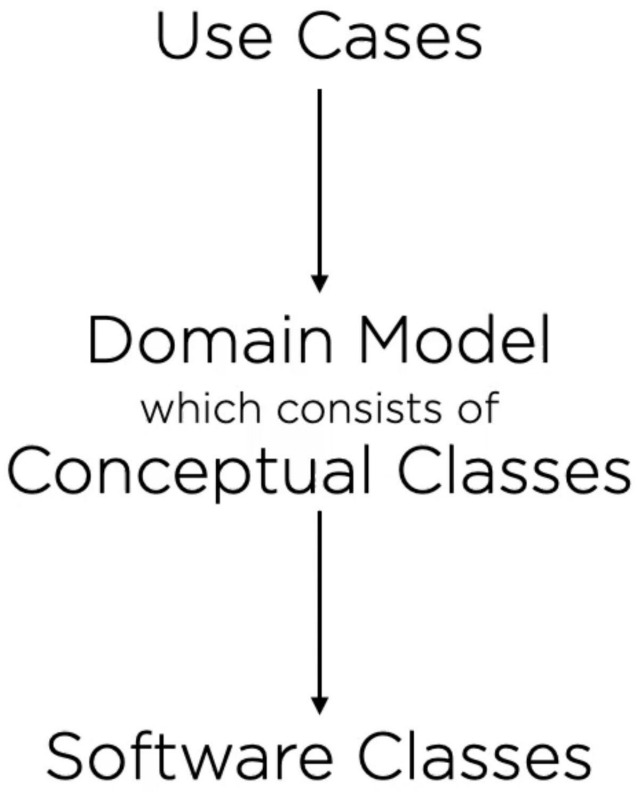
"A customer creates an order by first adding various products (digital or physical) to a shopping cart, then checks out, making a payment using a credit card."

FULFILL ORDER



"For an order, the system discovers which distribution centers hold stock of the products in the order. Each centre which can help fulfill the order is sent details of the products required together with the customer's details."

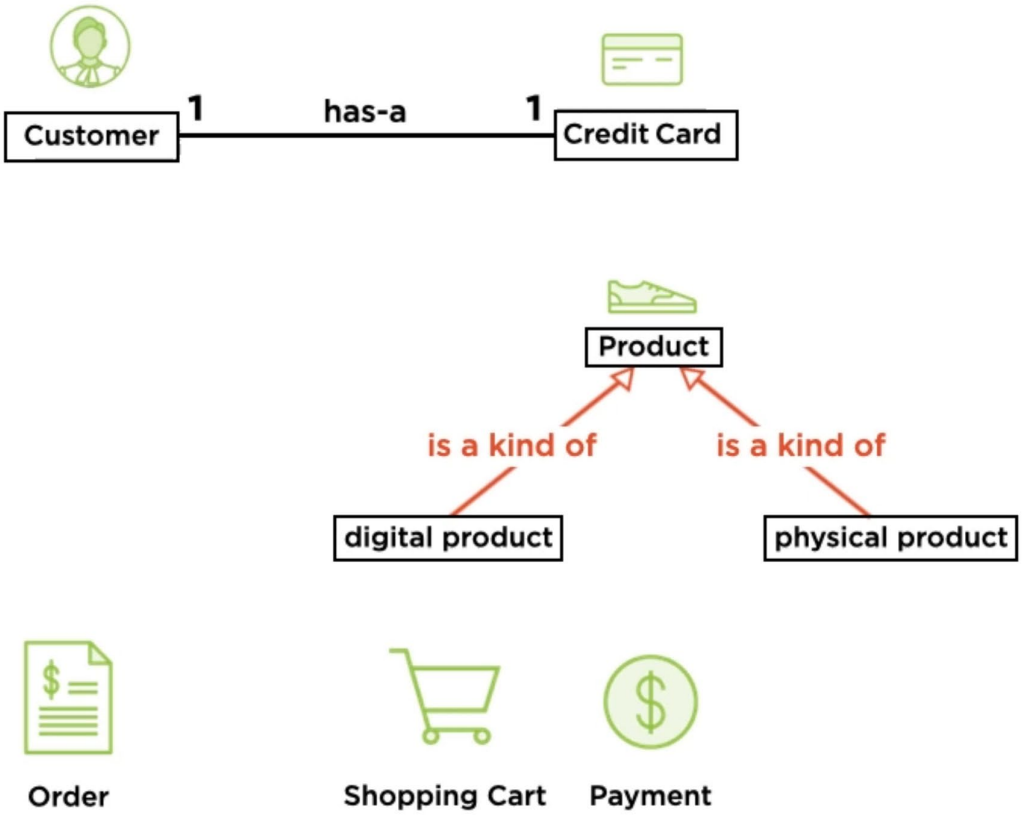
USE CASES


Use Cases
Domain Model
which consists of
Conceptual Classes
Software Classes

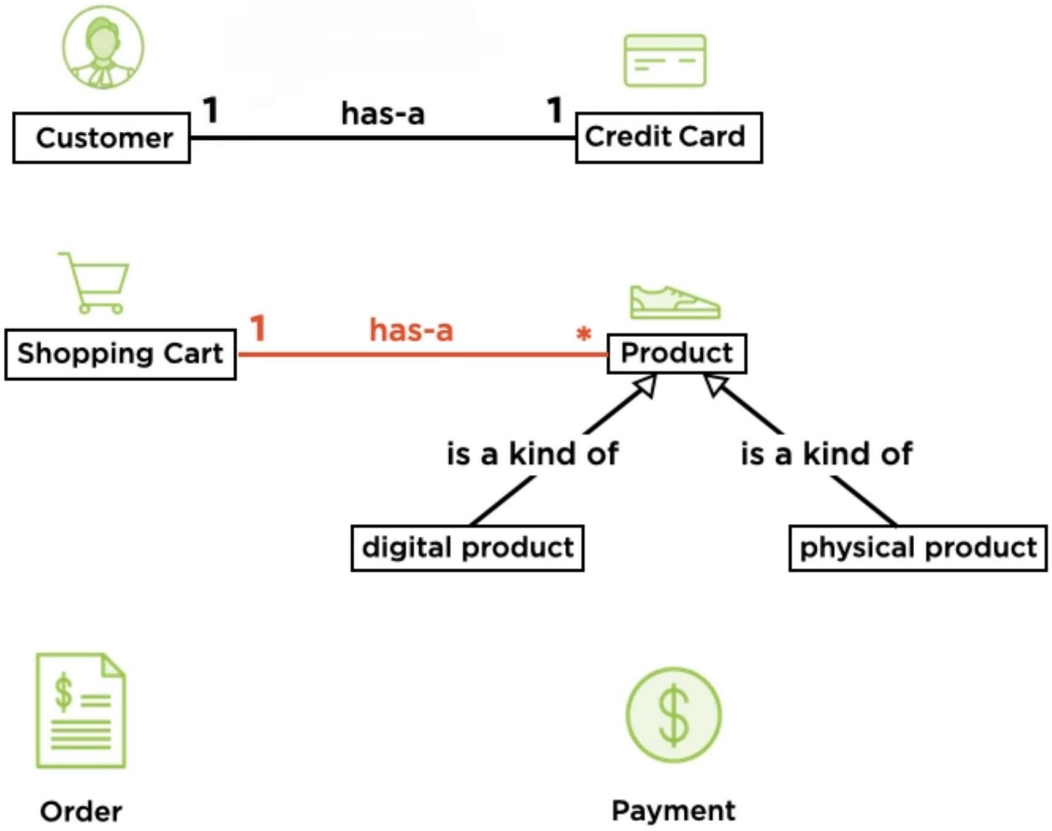
DISCOVERING CONCEPTUAL CLASSES

"A ***customer*** creates an ***order*** by first adding various ***products*** (digital or physical) to a ***shopping cart***, then checks out, making a ***payment*** using a ***credit card***."

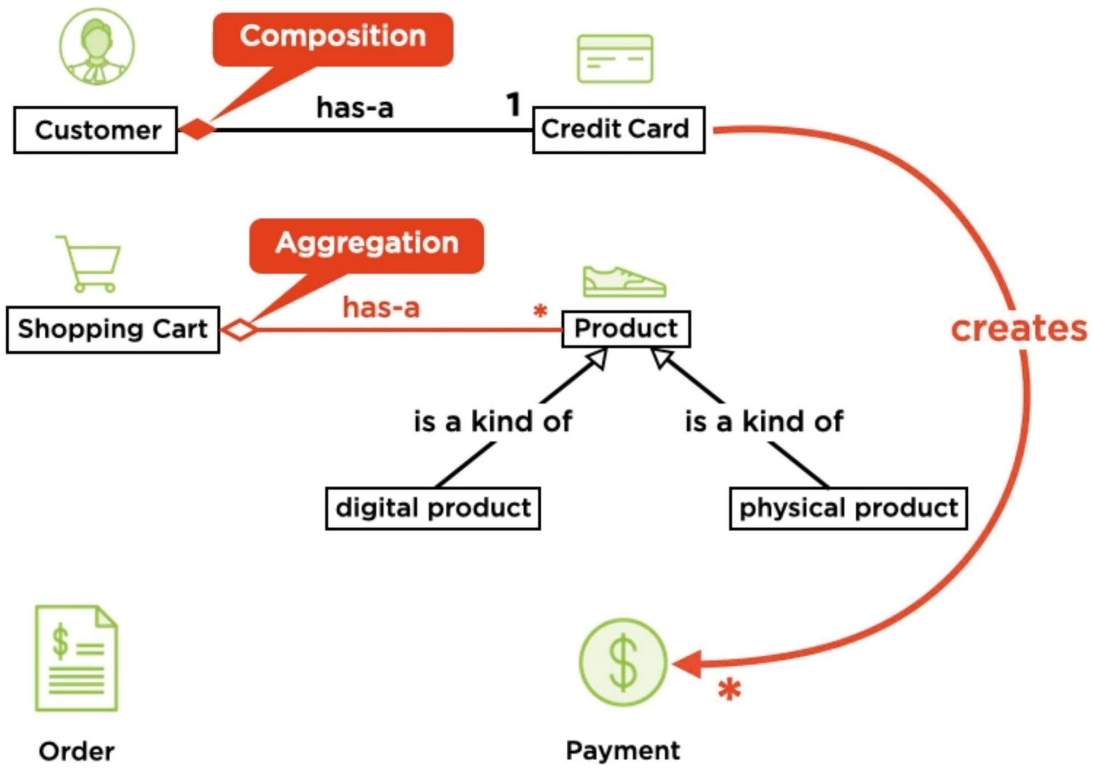
CLASS RELATIONS: HAS-A



CLASS RELATIONS

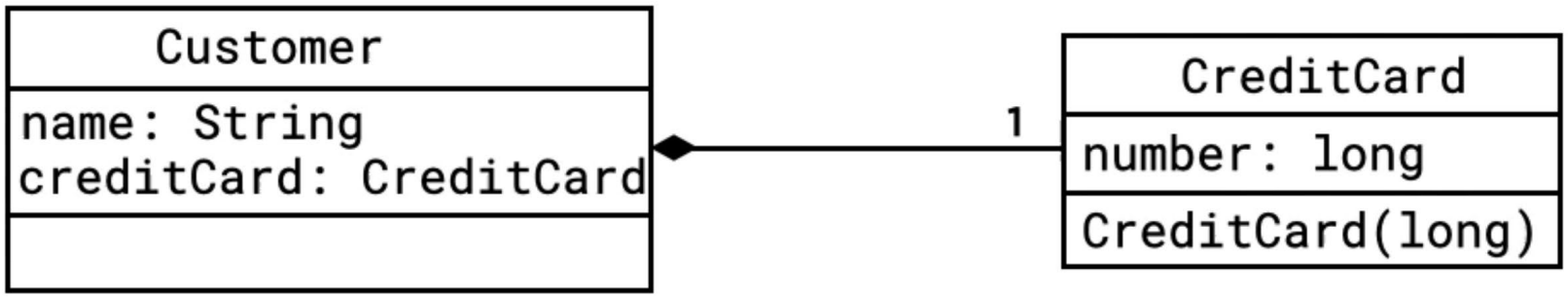


CLASS RELATIONS



SOFTWARE CLASSES

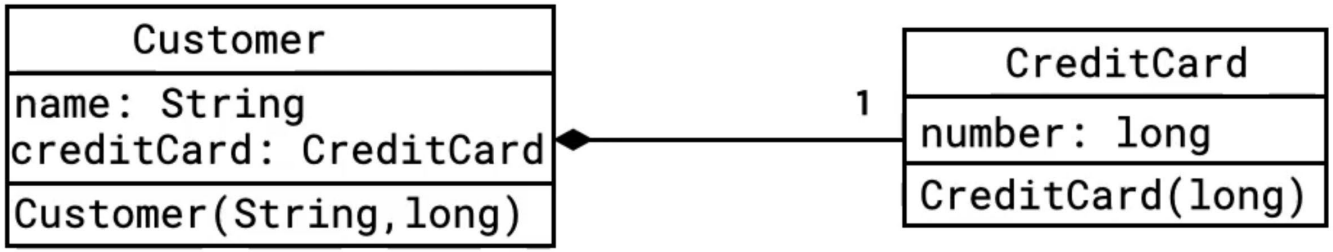
"A customer ... checks out, making a payment using a credit card."



SOFTWARE CLASSES

```java
public class Customer {

    private final String name;
    private CreditCard creditCard;

    public Customer(String name,
                    long ccNumber) {
        this.name = name;
        this.creditCard =
            new CreditCard(ccNumber);
    }
}
```

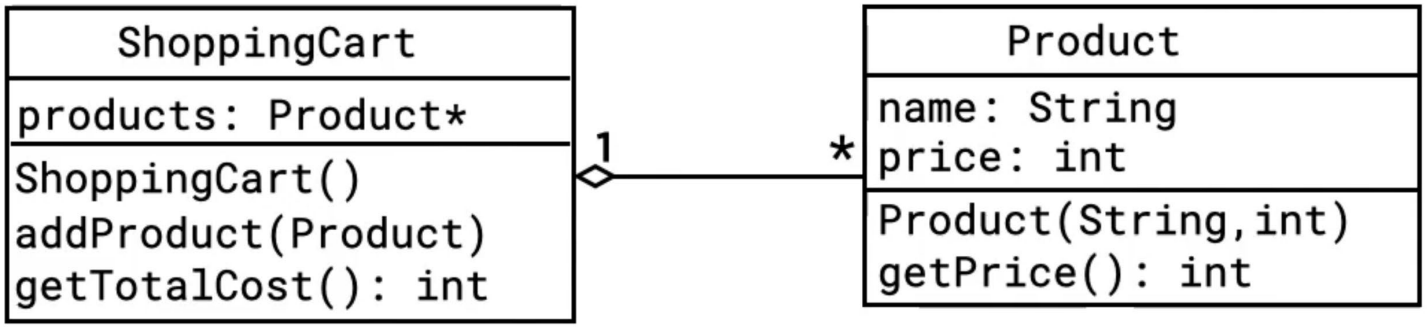
```java
class CreditCard {

    private final long cardNumber;

    CreditCard(long cardNumber) {
        this.cardNumber = cardNumber;
    }
}
```



SOFTWARE CLASSES

```java
public class ShoppingCart {

    private List<Product> products = new ArrayList<>();

    public void addProduct(Product product) {
        products.add(product);
    }

    public int getTotalCost() {
        return products.stream()
            .mapToInt(Product::getPrice)
            .sum();
    }
}
```

```java
public class Product {

    private final String name;
    private int price;

    public Product(String name,
                   int price) {
        this.name = name;
        this.price = price;
    }

    public int getPrice() {
        return price;
    }
}
```



QUESTIONS ?