|  |
| --- |
| **Set 23: Inheritance** |

|  |
| --- |
| **Skill 23.01: Explain the purpose of inheritance**  **Skill 23.02: Implement inheritance using the keyword extends**  **Skill 23.03: Implement the constructor of a super class using the keyword super**  **Skill 23.04: Invoke the methods of a super class in a subclass**  **Skill 23.05: Create objects from super and sub classes**  **Skill 23.06: Pass a sub class object as a parameter** |

|  |
| --- |
| **Skill 23.01: Explain the purpose of inheritance** |

**Skill 23.01 Concepts**

**Inheritance** is an important pillar of OOP(Object Oriented Programming). It is the mechanism in java by which one class is allow to inherit the features(fields and methods) of another class.

Important Terminology

* **Super Class**: The class whose features are inherited is known as super class(or a base class or a parent class).
* **Sub Class**: The class that inherits the other class is known as sub class(or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.
* **Reusability**: Inheritance supports the concept of “reusability”, i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

The short video below provides an overview of this concept

|  |
| --- |
|  |
| <https://www.youtube.com/watch?v=5nUPGlN2Ovo> |

|  |
| --- |
| **Skill 23.02: Implement inheritance using the keyword extends** |

**Skill 23.02 Concepts**

The keyword used for inheritance is *extends*.

The code snippet below illustrates one method for declaring, initializing, and populating an array. This method is only useful if we know the identity of the elements we want to store.

|  |
| --- |
| class derived-class extends base-class  {       //methods and fields  } |

In the below example of inheritance, class *Bicycle* is a base class, class *MountainBike* is a derived class which extends Bicycle class and class Main is a driver class to run program.

|  |
| --- |
| **Bicycle** |
| class Bicycle  {    public int gear;    public int speed;      public Bicycle(int gear, int speed)    {        this.gear = gear;        this.speed = speed;    }  } |
| **MountainBike** |
| class MountainBike extends Bicycle  {    //the MountainBike subclass adds one more field    public int seatHeight;      // the MountainBike subclass has one constructor    public MountainBike(int startHeight, int g, int s)    {          // invoking base-class(Bicycle) constructor          super(g, s);          seatHeight = startHeight;  }  } |

In the above example, the *Bicycle* class is referred to as our superclass. The *MountainBike* class is referred to as our subclass. By using the work "extends" in the *MountainBike* class, the *MountainBike* class inherits all the methods and state variables of the *Bicycle* class.

[**Skill 23.02: Exercise 1**](file:///C:\Users\PLUSKH01\Desktop\APCompSciA\ticketOutTheDoor\set19\Set19TicketOutTheDoorAPCompSciA.pdf)

|  |
| --- |
| **Skill 23.03: Implement the constructor of a super class using the keyword super** |

**Skill 23.03 Concepts**

As previously mentioned, the keyword **extends** causes a subclass to inherit all the methods and state variables of its super class. This is also true for the constructor.

To invoke the constructor of a super class the keyword **super** is used. This concept is illustrated below.

|  |
| --- |
| **The use of super to invoke a super class** |
| A screenshot of a computer  Description automatically generated |

There are some significant features of inheritance illustrated above. In the absence of *super(gear, speed)*, the *MountainBike* constructor would have tried to automatically call the Bicycle constructor and would have failed, since the Bicycle constructor requires two parameters and we would not have supplied these. By making *super(gear, speed)* the first line of code, we are able to supply the needed parameters. When used, *super()* must be the first line of code in the constructor of the subclass.

[**Skill 23.03: Exercises 1**](file:///C:\Users\PLUSKH01\Desktop\APCompSciA\ticketOutTheDoor\set19\Set19TicketOutTheDoorAPCompSciA.pdf)

|  |
| --- |
| **Skill 23.04: Invoke the methods of a super class in a subclass** |

**Skill 23.04 Concepts**

Because the *MountainBike* class extends the Bicycle class, *MountainBike* is able to access all the public methods and variables associated with Bicycle. This concept is illustrated below,

|  |
| --- |
| **Invoking the methods of a super class** |
| A screen shot of a computer program  Description automatically generated |

[**Skill 23.04: Exercises 1**](file:///C:\Users\PLUSKH01\Desktop\APCompSciA\ticketOutTheDoor\set19\Set19TicketOutTheDoorAPCompSciA.pdf)

|  |
| --- |
| **Skill 23.05: Create objects from super and sub classes** |

**Skill 23.05 Concepts**

The inheritance of one class from another follows an "IS A" relationship. That is, a mountain bike "IS A" bicycle. The reverse is not true, however. For example, a bike is not necessarily a mountain bike. When creating objects from super and sub classes, this relationship becomes important.

Consider the following hierarchy of inherited classes between bicycles.

A diagram of a bicycle

Description automatically generated

The above hierarchy shows the relationship among the classes in a program. According to the hierarchy

* CrossCountryBike "IS A" MountainBike
* DownhillBike "IS A" MountainBike
* MountainBike "IS A" Bicycle

When creating objects from super and sub classes this relationship is enforced. This is illustrated below,

A blue arrows with red x in the middle

Description automatically generated

[**Skill 23.05: Exercises 1**](file:///C:\Users\PLUSKH01\Desktop\APCompSciA\ticketOutTheDoor\set19\Set19TicketOutTheDoorAPCompSciA.pdf)

|  |
| --- |
| **Skill 23.06: Pass a sub class object as a parameter** |

**Skill 23.06 Concepts**

Any time when a parameter is expecting to receive an object of a particular type, it is acceptable to send it an object of the same class or a subclass, but never of a superclass. This is because the passed class object inherits all the methods of the object. If you attempt to pass an object of a parent class, the expected object may not have all the expected methods. Consider the following hierarchy of classes where each class is a subclass of the class immediately above it.

A screen shot of a diagram

Description automatically generated

Suppose there is a method with the following signature,



The method *theMethod* is clearly expecting a *Male* object; therefore, the following calls to this method would be legal since we are either sending a Male object or an object of a subclass,

A close-up of a white background

Description automatically generated

Since *theMethod* is expecting a *Male* object, we can’t send an object of a superclass.

A close-up of a grey background

Description automatically generated

[**Skill 23.06: Exercises 1**](file:///C:\Users\PLUSKH01\Desktop\APCompSciA\ticketOutTheDoor\set19\Set19TicketOutTheDoorAPCompSciA.pdf)