In previous part Person was the base class which is being extended by the Athlete class which is further extended by Sport classes meaning each sport player is an athlete. Each child classes i.e. sport player classes had few common methods e.g. doThis() method is common. However, in this part, the common methods have been moved to base class i.e. doThis() and Data() methods are common to all classes hence they are the part of Person class, and in order to enforce implementation of these methods in child classes these methods are marked as abstract (doThis() is abstract and Data() method is from PlayerData interface).

Additionally, the Person class is marked as Abstract. These changes brings a lot of benefits in terms of polymorphism, as we can see now we can create an array of 100 Person classes and we can assign different type of sport player to array locations since each sport player is a Person. Hence, we just to create array with the base class name to store different type of sport player. Moreover, since common methods are available in the base class, we can access those methods using the base class reference. This feature brings lot of benefit in terms of scalability of the system. Anytime we can add new sport player without changing the existing classes

**PlayerData.Java**

//abstract are defualt

public interface PlayerData {

void Data();

}

**Person.Java**

public abstract class Person implements PlayerData {

private String firstName;

private String middleName;

private String lastName;

private int age;

public Person(String firstName, String middleName, String lastName, int age) {

this.firstName = firstName;

this.middleName = middleName;

this.lastName = lastName;

this.age = age;

}

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getMiddleName() {

return middleName;

}

public void setMiddleName(String middleName) {

this.middleName = middleName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

public abstract void doThis();

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (obj == null)

return false;

if (getClass() != obj.getClass())

return false;

Person other = (Person) obj;

if (age != other.age)

return false;

if (firstName == null) {

if (other.firstName != null)

return false;

} else if (!firstName.equals(other.firstName))

return false;

if (lastName == null) {

if (other.lastName != null)

return false;

} else if (!lastName.equals(other.lastName))

return false;

if (middleName == null) {

if (other.middleName != null)

return false;

} else if (!middleName.equals(other.middleName))

return false;

return true;

}

//this method constructs a string for the info for each athlete

public String toString() {

return "\nFirst Name: " + firstName + "\nMiddle Name: " + middleName + "\nLast Name: " + lastName + " \nAge: " + age

+ "\n";

}

}

**Athlete.Java**

// Athlete is a person hence extends Person class

public class Athlete extends Person {

// Athlete has team and position

private String team;

private String position;

//constructor

public Athlete(String firstName, String middleName, String lastName, int age, String team, String position) {

super(firstName, middleName, lastName, age);

this.team = team;

this.position = position;

}

// This will override the abstract method doThis() from Person class

@Override

public void doThis() {

System.out.println("I enjoying Playing");

}

//accessors & mutators

public String getTeam() {

return team;

}

public void setTeam(String team) {

this.team = team;

}

public String getPosition() {

return position;

}

public void setPosition(String position) {

this.position = position;

}

// Declaring the equal method for Athlete

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (!(obj instanceof Athlete))

return false;

Athlete other = (Athlete) obj;

if (position == null) {

if (other.position != null)

return false;

} else if (!position.equals(other.position))

return false;

if (team == null) {

if (other.team != null)

return false;

} else if (!team.equals(other.team))

return false;

return true;

}

// This will override the abstract method Data() from PlayerData interface

@Override

public void Data() {

}

//this method constructs a string for the data for each athlete

@Override

public String toString() {

return super.toString() + "Team: " + team + "\nPosition: " + position + "\n";

}

}

**BadmintonPlayer.Java**

import java.util.Objects;

//player is an athlete and a person too hence extends both

public class BadmintonPlayer extends Athlete {

//Player has a shuttle cock

//players either left side or right side.

private String shuttleCock;

public BadmintonPlayer(String firstName, String middleName, String lastName, int age, String team, String position,

String shuttleCock) {

super(firstName, middleName, lastName, age, team, position);

this.shuttleCock = shuttleCock;

}

public String getShuttleCock() {

return shuttleCock;

}

public void setShuttleCock(String shuttleCock) {

this.shuttleCock = shuttleCock;

}

// this method constructs equal method for the Data for each athlete

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (!(obj instanceof BadmintonPlayer))

return false;

BadmintonPlayer other = (BadmintonPlayer) obj;

return Objects.equals(shuttleCock, other.shuttleCock);

}

// This will override the abstract method doThis() from Person class

public void doThis() {

System.out.println("I hit the shuttlecock");

}

// This will override the abstract method PlayerData() from Data interface

@Override

public void Data() {

}

public String toString() {

String Data;

Data = "Badminton Player:\n" + super.toString() + "Shuttle Cock: " + shuttleCock + "\n";

return Data;

}

}

**BaseballPlayer.Java**

import java.util.Objects;

// Baseball player is an athlete and a person too hence extends both

public class BaseballPlayer extends Athlete {

// Baseball Player has batting position

// Baseball players either bat lefthanded, righthanded or both.

private String battingPosition;

// Declaring constructors

public BaseballPlayer(String firstName, String middleName, String lastName, int age, String team, String position,

String battingPosition) {

super(firstName, middleName, lastName, age, team, position);

this.battingPosition = battingPosition;

}

public String getBattingPosition() {

return battingPosition;

}

public void setBattingPosition(String battingPosition) {

this.battingPosition = battingPosition;

}

// This will override the abstract method doThis() from Person

@Override

public void doThis() {

System.out.println("I hit something");

}

// Overriding equals() to compare two objects

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (!(obj instanceof BaseballPlayer))

return false;

BaseballPlayer other = (BaseballPlayer) obj;

return Objects.equals(battingPosition, other.battingPosition);

}

// This will override the abstract method Data() from PlayerData interface

public void Data() {

}

// this method constructs a tostring for the Data for each Baseball Player

public String toString() {

String Data;

Data = "\nBaseball Player:\n" + super.toString() + "Batting Position: " + battingPosition + "\n";

return Data;

}

}

**FootballPlayer.Java**

//Importing player array

import java.util.Arrays;

//Football player is an athlete and a person too hence extends both

public class FootballPlayer extends Athlete {

// Player may play defense or offense

// player may play special teams

private String speciality[];

public FootballPlayer(String firstName, String middleName, String lastName, int age, String team, String position,

String[] speciality) {

super(firstName, middleName, lastName, age, team, position);

this.speciality = speciality;

}

public String[] getSpeciality() {

return speciality;

}

public void setSpeciality(String[] speciality) {

this.speciality = speciality;

}

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (!(obj instanceof FootballPlayer))

return false;

FootballPlayer other = (FootballPlayer) obj;

return Arrays.equals(speciality, other.speciality);

}

// This will override the abstract method doThis() from Person class

@Override

public void doThis() {

System.out.println("I tackle something");

}

// This will override the abstract method PlayerData() from Data interface

public void Data() {

//This method constructs a string for the player data

}

@Override

public String toString() {

String Data;

Data = "Football Player:\n" + super.toString() + "Speciality: " + Arrays.toString(speciality) + "\n";

return Data;

}

}

**Golfer.Java**

import java.util.Objects;

// Golfer player is an athlete and a person too hence extends both

public class Golfer extends Athlete {

// Golfer has a sponsor

private String sponsor;

public Golfer(String firstName, String middleName, String lastName, int age, String team, String position,

String sponsor) {

super(firstName, middleName, lastName, age, team, position);

this.sponsor = sponsor;

}

public String getSponsor() {

return sponsor;

}

public void setSponsor(String sponsor) {

this.sponsor = sponsor;

}

// this method constructs equal method for the Data for each athlete

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (!(obj instanceof Golfer))

return false;

Golfer other = (Golfer) obj;

return Objects.equals(sponsor, other.sponsor);

}

// This will override the abstract method doThis() from Person class

public void doThis() {

System.out.println("I putt in the hole");

}

// This will override the abstract method Data() from PlayerData interface

public void Data() {

}

// this method constructs a string for the Data for each Golfer

public String toString() {

String Data;

Data = "Golfer:\n" + super.toString() + "Sponsor: " + sponsor + "\n";

return Data;

}

}

**HockeyPlayer.Java**

import java.util.Objects;

// Hockey player is an athlete and a person too hence extends both

public class HockeyPlayer extends Athlete {

// Hockey player has stickBand

private String stickBrand;

public HockeyPlayer(String firstName, String middleName, String lastName, int age, String team, String position,

String stickBrand) {

super(firstName, middleName, lastName, age, team, position);

this.stickBrand = stickBrand;

}

public String getStickBrand() {

return stickBrand;

}

public void setStickBrand(String stickBrand) {

this.stickBrand = stickBrand;

}

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (!(obj instanceof HockeyPlayer))

return false;

HockeyPlayer other = (HockeyPlayer) obj;

return Objects.equals(stickBrand, other.stickBrand);

}

// This will override the abstract method doThis() from Person class

public void doThis() {

System.out.println("I sit in a penalty box");

}

// This will override the abstract method Data() from PlayerData interface

public void Data() {

}

// this method constructs a string for the Data for each Hockey Player

public String toString() {

String Data;

Data = "Hockey Player:\n" + super.toString() + "Stick Brand: " + stickBrand + "\n";

return Data;

}

}

**HandballPlayer.Java**

import java.util.Objects;

//player is an athlete and a person too hence extends both

public class HandballPlayer extends Athlete {

// player has handball gloves

// players play centerback only

private String handballGloves;

public HandballPlayer(String firstName, String middleName, String lastName, int age, String team, String position,

String handballGloves) {

super(firstName, middleName, lastName, age, team, position);

this.handballGloves = handballGloves;

}

public String getHandballGloves() {

return handballGloves;

}

public void setHandballGloves(String handballGloves) {

this.handballGloves = handballGloves;

}

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (!(obj instanceof HandballPlayer))

return false;

HandballPlayer other = (HandballPlayer) obj;

if (this.getAge() != other.getAge())

return false;

return Objects.equals(handballGloves, other.handballGloves);

}

// This will override the abstract method doThis() from Person class

public void doThis() {

System.out.println("I throw the ball");

}

// This will override the abstract method PlayerData() from Data interface

@Override

public void Data() {

}

public String toString() {

String Data;

Data = "Handball Player:\n" + super.toString() + "Handball Gloves: " + handballGloves + "\n";

return Data;

}

}

**RugbyPlayer.Java**

import java.util.Objects;

//Rugby player is an athlete and a person too hence extends both

public class RugbyPlayer extends Athlete {

// Player has cleats brand

//Leftwing, Fullback or Both

private String cleatsBrand;

public RugbyPlayer(String firstName, String middleName, String lastName, int age, String team, String position,

String cleatsBrand) {

super(firstName, middleName, lastName, age, team, position);

this.cleatsBrand = cleatsBrand;

}

public String getCleatsBrand() {

return cleatsBrand;

}

public void setCleatsBrand(String cleatsBrand) {

this.cleatsBrand = cleatsBrand;

}

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (!(obj instanceof RugbyPlayer))

return false;

RugbyPlayer other = (RugbyPlayer) obj;

return Objects.equals(cleatsBrand, other.cleatsBrand);

}

// This will override the abstract method doThis() from Person class

@Override

public void doThis() {

System.out.println("I catch the quanco");

}

// This will override the abstract method Data() from PlayerData interface

@Override

public void Data() {

}

// this method constructs a string for the Data for each Player

public String toString() {

String Data;

Data = "RugbyPlayer:\n" + super.toString() + "Cleats Brand: " + cleatsBrand + "\n";

return Data;

}

}

**SoccerPlayer.Java**

// Soccer player is an athlete and a person too hence extends both

public class SoccerPlayer extends Athlete {

// goalKeeper, defender, midFielder, or forward

private String fieldPosition;

public SoccerPlayer(String firstName, String middleName, String lastName, int age, String team, String position,

String fieldPosition) {

super(firstName, middleName, lastName, age, team, position);

this.fieldPosition = fieldPosition;

}

// do this method to display message

public String getFieldPosition() {

return fieldPosition;

}

public void setFieldPosition(String fieldPosition) {

this.fieldPosition = fieldPosition;

}

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (!(obj instanceof SoccerPlayer))

return false;

SoccerPlayer other = (SoccerPlayer) obj;

if (fieldPosition == null) {

if (other.fieldPosition != null)

return false;

} else if (!fieldPosition.equals(other.fieldPosition))

return false;

return true;

}

// This will override the abstract method doThis() from Person class

public void doThis() {

System.out.println("I kick the ball");

}

// This will override the abstract method Data() from PlayerData interface

public void Data() {

}

// This method constructs a string for the player data

@Override

public String toString() {

String Data;

Data = "Soccer Player:\n" + super.toString() + "Field Position: " + fieldPosition + "\n";

System.out.println(Data);

}

}

**TennisPlayer.Java**

import java.util.Objects;

//Tennis player is an athlete and a person too hence extends both

public class TennisPlayer extends Athlete {

// Tennis player has a RacketBrand

// Player i has a position rightcorner or leftcorner or both

private String racketBrand;

public TennisPlayer(String firstName, String middleName, String lastName, int age, String team, String position,

String racket) {

super(firstName, middleName, lastName, age, team, position);

this.racketBrand = racket;

}

public String getRacket() {

return racketBrand;

}

public void setRacket(String racketBrand) {

this.racketBrand = racketBrand;

}

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (!(obj instanceof TennisPlayer))

return false;

TennisPlayer other = (TennisPlayer) obj;

if (this.getAge() != other.getAge())

return false;

return Objects.equals(racketBrand, other.racketBrand);

}

// This will override the abstract method doThis() from Person class

public void doThis() {

System.out.println("I hit the net");

}

// This will override the abstract method Data() from PlayerData interface

public void Data() {

}

@Override

public String toString() {

String Data;

Data = "Tennis Player:\n" + super.toString()+ "Racket Brand: " + racketBrand + "\n";

return Data;

}

}

**TugOfWarPlayer.Java**

import java.util.Objects;

//player is an athlete and a person too hence extends both

public class TugOfWarPlayer extends Athlete {

// player has sturdy footwear

// player may play back or front not both

private String sturdyFootWear;

public TugOfWarPlayer(String firstName, String middleName, String lastName, int age, String team, String position,

String sturdyFootWear) {

super(firstName, middleName, lastName, age, team, position);

this.sturdyFootWear = sturdyFootWear;

}

public String getSturdyFootWear() {

return sturdyFootWear;

}

public void setSturdyFootWear(String sturdyFootWear) {

this.sturdyFootWear = sturdyFootWear;

}

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (!(obj instanceof TugOfWarPlayer))

return false;

TugOfWarPlayer other = (TugOfWarPlayer) obj;

if (this.getAge() != other.getAge())

return false;

return Objects.equals(sturdyFootWear, other.sturdyFootWear);

}

// This will override the abstract method doThis() from Person class

public void doThis() {

System.out.println("I pull the rope");

}

// This will override the abstract method PlayerData() from Data interface

@Override

public void Data() {

}

public String toString() {

String Data;

Data = "Tug Of War Player:\n" + super.toString() + "Sturdy Footwear: " + sturdyFootWear + "\n";

return Data;

}

}

**VolleyballPlayer.Java**

import java.util.Objects;

// Volleyball player is an athlete and a person too hence extends both

public class VolleyballPlayer extends Athlete {

// Volleyball player has fingerBands

// Volleyball player players either Libero or Wing spiker or both

private String fingerBands;

public VolleyballPlayer(String firstName, String middleName, String lastName, int age, String team, String position,

String fingerBands) {

super(firstName, middleName, lastName, age, team, position);

this.fingerBands = fingerBands;

}

public String getFingerBands() {

return fingerBands;

}

public void setFingerBands(String fingerBands) {

this.fingerBands = fingerBands;

}

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (!(obj instanceof VolleyballPlayer))

return false;

VolleyballPlayer other = (VolleyballPlayer) obj;

return Objects.equals(fingerBands, other.fingerBands);

}

// This will override the abstract method doThis() from Person class

public void doThis() {

System.out.println("I spike the ball");

}

// This will override the abstract method PlayerData() from Data interface

@Override

public void Data() {

}

//this method constructs a string for the Data for each Player

public String toString() {

String Data;

Data = "Volleyball Player:\n" + super.toString() + "Finger Bands: " + fingerBands + "\n";

return Data;

}

}

**TestPlayer.Java**

import java.util.Scanner;

public class TestPlayer {

public static void main(String[] args) {

Person personArray[] = new Person[100];

// Creating objects

Person Hank = new BaseballPlayer("Hank", "A.", "Aaron", 29, "Orlando Magic", "Batter1","left-handed");

String s[]= {"Defense", "Pittsburgh Steelers"};

Person Terry = new FootballPlayer("Terry","F.", "Bradshaw", 30, " Los Angeles Rams ", "LeftWinger", s);

Person Mario = new HockeyPlayer("Mario", "R", "Lemieux", 53, "Pittsburgh Penguins", "Goalie", "CCW");

Person Paula = new Golfer("Paula", "X.", "Creamer", 32, "SkyGolf", "Unknown", "CDW");

Person Brain = new SoccerPlayer("Brain", "T.", "Rowe", 30, " Manchester City F.C ", "Number2", "Defender");

Person Barry = new BaseballPlayer("Barry", "B.", "Bonds", 40, "Pittsburgh Pirates", "Batter2","Both");

String speciality[] = {"Attacker", "Offense","Defense"};

Person Payton = new FootballPlayer("Payton", "A.", "Manning", 27, "Argentina", "CornerBack", speciality);

Person Marcus = new BadmintonPlayer("Marcus", "F.", "Gideon", 27, "Indonesia", "Right", "Carlton");

Person Wayne = new HockeyPlayer("Wayne", "D.", "Gretzky", 34, "New Jersey Devils", "Skater", "KTG");

Person Nikola = new HandballPlayer("Nikola", "M", "Karabatic", 20, "PSG", "Centre Back", "Deerskin");

Person Rogers = new TennisPlayer("Rogers", "F.", "Federer", 32, "Davis Cup ", "Both", "Babolat");

Person Phil = new Golfer("Phil", "A.", "Mickleson", 49, "All-American ", "Unknown", "KPMG");

Person Dustin = new VolleyballPlayer("Dustin", "P.", "Watten", 23, "SCC Berlin ", "Both", "Reebok");

Person Dom = new SoccerPlayer("Dom", "A", "Dwyer", 45, "Arsenel F.C ", "Number1", "GoalKeeper");

Person David = new RugbyPlayer("David", "S", "Pocock", 32, "Brumbies ", "Both", "Nike");

Person Maria = new TennisPlayer("Maria", "Y.", "Sharapova", 32, "Fed Cup ", "RightCorner", "Babolat");

Person Pierre = new RugbyPlayer("Pierre", "A.", "Spies", 35, "SpringBrook ", "LeftWing", "Adidas");

Person Ed = new TugOfWarPlayer("Ed", "S.", "Mill", 27, "Great Britain", "Front", "Reebok");

Person Taylor = new VolleyballPlayer("Taylor", "S.", "Sander", 27, "Associacao Social Esportiva","Wing spiker", "NBS");

Person Della = new BadmintonPlayer("Della", "D", "Haris", 34, "Jakarta", "Left", "Yonex");

Person Mikkel = new HandballPlayer("Mikkel", "M", "Hansen", 20, "PSG", "Centre Back", "Deerskin");

Person Edwin = new TugOfWarPlayer("Edwin", "S.", "Mill", 27, "Great Britain", "Back", "Reebok");

int index = 0;

personArray[index++] = Hank;

personArray[index++] = Terry;

personArray[index++] = Mario;

personArray[index++] = Paula;

personArray[index++] = Brain;

personArray[index++] = Barry;

personArray[index++] = David;

personArray[index++] = Maria;

personArray[index++] = Payton;

personArray[index++] = Wayne;

personArray[index++] = Phil;

personArray[index++] = Dustin;

personArray[index++] = Dom;

personArray[index++] = Marcus;

personArray[index++] = Rogers;

personArray[index++] = Pierre;

personArray[index++] = Taylor;

personArray[index++] = Edwin;

personArray[index++] = Mikkel;

personArray[index++] = Della;

personArray[index++] = Nikola;

personArray[index++] = Ed;

int choice = 0;

Scanner scanner = new Scanner(System.in);

do {

System.out.println("\nMain Menu. Welcome to Program Athlete");

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

System.out.println("1. By Sport, list the players that are equal to each other");

System.out.println("2. Call each sports doThis() method (one at a time) passing each player");

System.out.println("3. Call the toString methods for each player object");

System.out.println("0. Exit");

System.out.print("\nEnter your choice: ");

choice = scanner.nextInt();

//scanner.nextLine();

switch(choice) {

case 1:

for(int i = 0; i < index; i++) {

for(int j = i; j < index; j++) {

If(i != j) {

if(personArray[i].equals(personArray[j])) {

System.out.println(personArray[i].getFirstName() + " and " + personArray[j].getFirstName() + " are equal");

}

}

}

}

break;

case 2:

for(int i = 0; i < index; i++) {

personArray[i].doThis();

}

break;

case 3:

for(int i = 0; i < index; i++) {

System.out.println(personArray[i].toString());

}

break;

case 0:

System.out.println("Exiting Menu");

break;

default:

break;

}

}while(choice != 0);

scanner.close();

}

}