11/14/22

Replit Notes: Accessor and Mutator Methods

- Avoid duplicating the same code in the constructor and the setter
  - Make a new Method with a single copy of the code (more on that today!)
- Be careful when refactoring existing code that you do not break the existing assumptions
  - `Student.fullName` always equals `Student.firstName + " " + Student.lastName`
  - Mutator methods for `Student.firstName` and/or `Student.lastName` are required to regenerate `Student.fullName`
- Do not provide Mutable Methods if they are not needed
  - No need to provide a setter for `Contact.areaCode`
- Consider removing `Student.fullName` and `Contact.areaCode` instance variables and generate these values on-demand from the related components
  - This could save a ton of space (although speed may also need to be considered) and removes the need to keep multiple things in sync when something changes

5.6: Writing Methods

Methods

- We have already covered about HOW to create Methods - but we have not spent much time talking about WHEN you should consider moving code into a Method
- Some of the WHENs
  - You have the same (or very nearly the same) block of code written in multiple places
  - You want to reduce complexity (improve development velocity / reduce code brittleness)
  - You want to write tests for a block of code
  - You have Methods that are excessively long (more than a single page)

Methods: Repeated Code

```
public class Person {

  private String firstName;
  private String lastName;

  public Person(String fn, String ln) {
    firstName = fn;
    lastName = ln;
  }

  public void sayHello() {
    String fullName = firstName + " " + lastName;
    System.out.println("Hello " + fullName);
  }

  public void sayGoodbye() {
    String fullName = firstName + " " + lastName;
    System.out.println("Goodbye " + fullName);
  }
}
```

Methods: Repeated Code

```java
public class Person {

  private String firstName;
  private String lastName;

  public Person(String fn, String ln) {
    firstName = fn;
    lastName = ln;
  }

  public void sayHello() {
    String fullName = firstName + " " + lastName;
    System.out.println("Hello " + fullName);
  }

  public void sayGoodbye() {
    String fullName = firstName + " " + lastName;
    System.out.println("Goodbye " + fullName);
  }
}
```

```java
public class Person {

  private String firstName;
  private String lastName;

  public Person(String fn, String ln) {
    firstName = fn;
    lastName = ln;
  }

  public void sayHello() {
    System.out.println("Hello " +  fullName());
  }

  public void sayGoodbye() {
    System.out.println("Goodbye " +  fullName());
  }

  private String fullName() {
    return firstName + " " + lastName;
  }
}
```

Methods: Repeated Code

```
public class Person {

  private String firstName;
  private String lastName;

  public Person(String fn, String ln) {
    fi
    las
  }

  publi
    St                                    tName;
    Sys                                   ;
  }

  public void sayGoodbye() {
    String fullName = firstName + " " + lastName;
    System.out.println("Goodbye " + fullName);
  }
}
```

And now it is super easy to add new functionality like...

```
public class Person {

  private String firstName;
  private String lastName;

  public Person(String fn, String ln) {
    firstName = fn;
    lastName = ln;
  }

  public void sayHello() {
    System.out.println("Hello " +  fullName());
  }

  public void sayGoodbye() {
    System.out.println("Goodbye " +  fullName());
  }

  private String fullName() {
    return firstName + " " + lastName;
  }
}
```

Methods: Repeated Code

```
public class Person {

  private String firstName;
  private String lastName;

  public Person(String fn, String ln) {
    fi
    las
  }

  publi
    St                                  tName;
    Sys                                 ;
  }

  public void sayGoodbye() {
    String fullName = firstName + " " + lastName;
    System.out.println("Goodbye " + fullName);
  }
}
```

And now it is super easy to add new functionality like...

```
public class Person {

  private String firstName;
  private String lastName;
  private String middleName;

  public Person(String fn, String mn, String ln) {
    firstName = fn;
    middleName = mn;
    lastName = ln;
  }

  public void sayHello() {
    System.out.println("Hello " +fullName());
  }

  public void sayGoodbye() {
    System.out.println("Goodbye " +fullName());
  }

  private String fullName() {
    return firstName + " " + middleName + " " + lastName;
  }
}
```

Methods: Repeated Code

```
public class Person {

  private String firstName;
  private String lastName;

  public Person(String fn, String ln) {
    fi
    las
  }

  publi
    St                                  tName;
    Sys                                 ;
  }

  public void sayGoodbye() {
    String fullName = firstName + " " + lastName;
    System.out.println("Goodbye " + fullName);
  }
}
```

...and existing code
"just works"

```
public class Person {

  private String firstName;
  private String lastName;
  private String middleName;

  public Person(String fn, String mn, String ln) {
    firstName = fn;
    middleName = mn;
    lastName = ln;
  }

  public void sayHello() {
    System.out.println("Hello " +fullName());
  }

  public void sayGoodbye() {
    System.out.println("Goodbye " +fullName());
  }

  private String fullName() {
    return firstName + " " + middleName + " " + lastName;
  }
}
```

Methods: Reduce Complexity

```
public class FenceMaintenance {
 private int fenceWidth, fenceHeight;
 public FenceMaintenace(int width, int height) {
   fenceWidth = width; fenceHeight = height;
 }
 public paintFence() {
   int paintBucketsNeeded = (fenceWidth * fenceHeight) /
10;
   int brushesNeeded = paintBucketsNeeded * 1.5;
   double totalCost = 1.50 * brushesNeeded;
   totalCost += 12.99 * paintBucketsNeeded;
   System.out.println("Supplies purchased for $" +
totalCost);
   int totalTime = fenceArea / 5;
   System.out.println("Time required is " + totalTime +
"minutes.");
 }
}
```

Methods: Reduce Complexity

```
public class FenceMaintenance {
 private int fenceWidth, fenceHeight;
 public FenceMaintenance(int width, int height) {
   fenceWidth = width; fenceHeight = height;
 }
 public paintFence() {
   int paintBucketsNeeded = (fenceWidth * fenceHeight) /
10;
   int brushesNeeded = paintBucketsNeeded * 1.5;
   double totalCost = 1.50 * brushesNeeded;
   totalCost += 12.99 * paintBucketsNeeded;
   System.out.println("Supplies purchased for $" +
totalCost);
   int totalTime = fenceArea / 5;
   System.out.println("Time required is " + totalTime +
"minutes.");
 }
}
```

Question: What are some of the things we can break out into their own Methods?

Methods: Reduce Complexity

```
public class FenceMaintenance {
 private int fenceWidth, fenceHeight;
 public FenceMaintenance(int width, int height) {
   fenceWidth = width; fenceHeight = height;
 }
 public paintFence() {
   int paintBucketsNeeded = (fenceWidth * fenceHeight) /
10;
   int brushesNeeded = paintBucketsNeeded * 1.5;
   double totalCost = 1.50 * brushesNeeded;
   totalCost += 12.99 * paintBucketsNeeded;
   System.out.println("Supplies purchased for $" +
totalCost);
   int totalTime = fenceArea / 5;
   System.out.println("Time required is " + totalTime +
"minutes.");
 }
}
```

```
public class FenceMaintenance {
  private int fenceWidth, fenceHeight;
  public FenceMaintenance(int width, int height) {
    fenceWidth = width; fenceHeight = height;
  }
  public paintFence() {
    System.out.println("Supplies purchased for $" + calcCosts());
    System.out.println("Time required is " + calcTimeNeeded() +
"minutes.");
  }
  private int calcPaintBucketsNeeded()
    return calcFenceArea() / 10;
  }
  private int calcFenceArea() {
    return fenceWidth * fenceHeight;
  private double calcCosts() {
    return calcBrushesNeeded()      * 1.50 +
           calcPaintBucketsNeeded() * 12.99;
  }
}
```

```
private int calcBrushesNeeded() {
  return calcPaintBucketsNeeded() *
1.5;
}
private int calcTimeNeeded() {
  return calcFenceArea() / 5;
}
```

Methods: Reduce Complexity

```
public class FenceMaintenance {
 private int fenceWidth, fenceHeight;
 public FenceMaintenance(int width, int height) {
   fenceWidth = width; fenceHeight = height;
 }
 public paintFence() {
                                        * fenceHeight)
                                        d * 1.5;
                                        ed;
                                        ed;
                                        d for $" +
totalCost);
   int totalTime = fenceArea / 5;
   System.out.println("Time required is " + totalTime +
"minutes.");
 }
}
```

Question: Now what are some things that can EASILY be added (or changed)?

```
public class FenceMaintenance {
  private int fenceWidth, fenceHeight;
  public FenceMaintenance(int width, int height) {
    fenceWidth = width; fenceHeight = height;
  }
  public paintFence() {
    System.out.println("Supplies purchased for $" + calcCosts());
    System.out.println("Time required is " + calcTimeNeeded() +
"minutes.");
  }
  private int calcPaintBucketsNeeded()
    return calcFenceArea() / 10;
  }
  private int calcFenceArea() {
    return fenceWidth * fenceHeight;
  private double calcCosts() {
    return calcBrushesNeeded()       * 1.50 +
           calcPaintBucketsNeeded() * 12.99;
  }
}
```

```
private int calcBrushesNeeded() {
  return calcPaintBucketsNeeded() *
1.5;
}
private int calcTimeNeeded() {
  return calcFenceArea() / 5;
}
```

Method Parameters: **Pass by Value** / Pass by Reference

- Parameters passed into Methods behave differently if they are primitive types or Object references
- Primitive Types (byte, short, int, long, float, double, boolean, char)
  - A copy is made when the Method is invoked for use during the life of the Method
  - The Method cannot alter the value of the variable (passed as a parameter) that exists in the caller of the Method

```
int age = 16;
Person p = new Person(age);
// age is guaranteed to still be 16

public class Person {
  public Person(int initAge) {
    // initAge is a COPY of age
    initAge = 20; // does not alter the value of age in the caller
  }
}
```

Method Parameters: **Pass by Value** / Pass by Reference

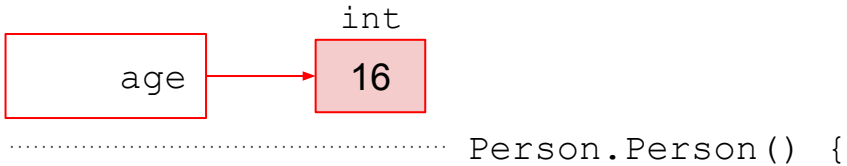
- Parameters passed into Methods behave differently if they are primitive types or Object references
- Primitive Types (byte, short, int, long, float, double, boolean, char)
  - A copy is made when the Method is invoked for use during the life of the Method
  - The Method cannot alter the value of the variable (passed as a parameter) that exists in the caller of the Method


```
int age = 16;
Person p = new Person(age);
// age is guaranteed to still be 16

public class Person {
  public Person(int initAge) {
    // initAge is a COPY of age
    initAge = 20; // does not alter the value of age in the caller
  }
}
```

Method Parameters: **Pass by Value** / Pass by Reference

- Parameters passed into Methods behave differently if they are primitive types or Object references
- Primitive Types (byte, short, int, long, float, double, boolean, char)
  - A copy is made when the Method is invoked for use during the life of the Method
  - The Method cannot alter the value of the variable (passed as a parameter) that exists in the caller of the Method

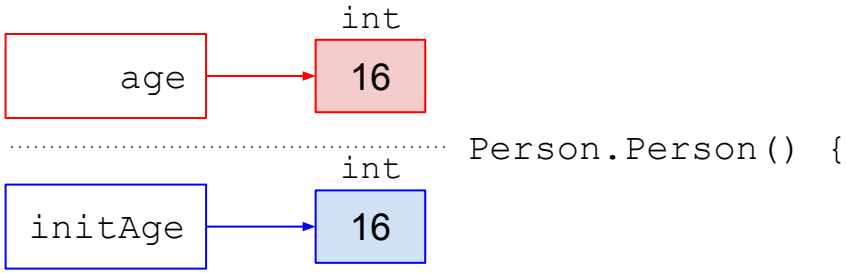
```
int age = 16;
Person p = new Person(age);
// age is guaranteed to still be 16

public class Person {
  public Person(int initAge) {
    // initAge is a COPY of age
    initAge = 20; // does not alter the value of age in the caller
  }
}
```



Method Parameters: **Pass by Value** / Pass by Reference

- Parameters passed into Methods behave differently if they are primitive types or Object references
- Primitive Types (byte, short, int, long, float, double, boolean, char)
  - A copy is made when the Method is invoked for use during the life of the Method
  - The Method cannot alter the value of the variable (passed as a parameter) that exists in the caller of the Method

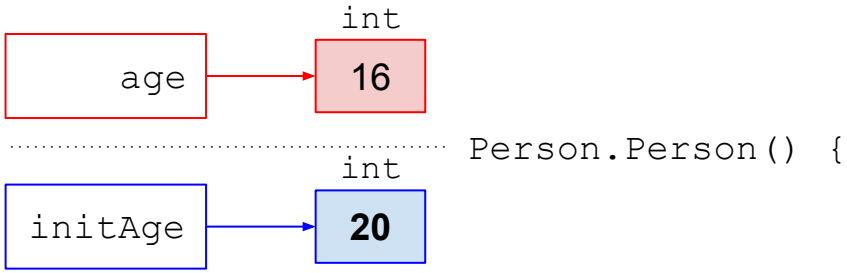
```
int age = 16;
Person p = new Person(age);
// age is guaranteed to still be 16

public class Person {
  public Person(int initAge) {
    // initAge is a COPY of age
    initAge = 20; // does not alter the value of age in the caller
  }
}
```



Method Parameters: **Pass by Value** / Pass by Reference

- Parameters passed into Methods behave differently if they are primitive types or Object references
- Primitive Types (byte, short, int, long, float, double, boolean, char)
  - A copy is made when the Method is invoked for use during the life of the Method
  - The Method cannot alter the value of the variable (passed as a parameter) that exists in the caller of the Method

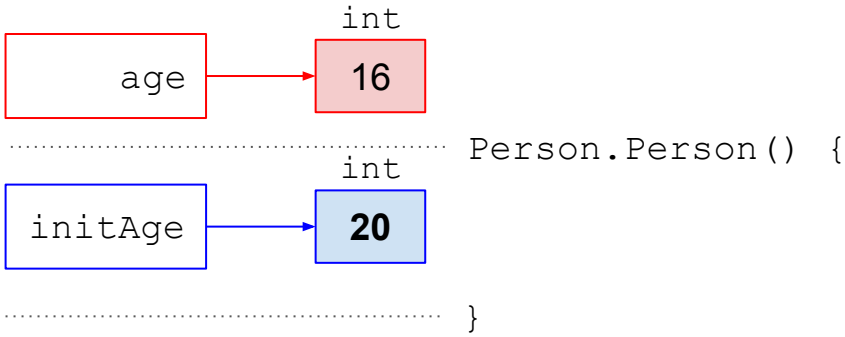
```
int age = 16;
Person p = new Person(age);
// age is guaranteed to still be 16

public class Person {
  public Person(int initAge) {
    // initAge is a COPY of age
    initAge = 20; // does not alter the value of age in the caller
  }
}
```



Method Parameters: **Pass by Value** / Pass by Reference

- Parameters passed into Methods behave differently if they are primitive types or Object references
- Primitive Types (byte, short, int, long, float, double, boolean, char)
  - A copy is made when the Method is invoked for use during the life of the Method
  - The Method cannot alter the value of the variable (passed as a parameter) that exists in the caller of the Method

```
int age = 16;
Person p = new Person(age);
// age is guaranteed to still be 16

public class Person {
  public Person(int initAge) {
    // initAge is a COPY of age
    initAge = 20; // does not alter the value of age in the caller
  }
}
```



Method Parameters: **Pass by Value** / Pass by Reference

- Parameters passed into Methods behave differently if they are primitive types or Object references
- Primitive Types (byte, short, int, long, float, double, boolean, char)
  - A copy is made when the Method is invoked for use during the life of the Method
  - The Method cannot alter the value of the variable (passed as a parameter) that exists in the caller of the Method


```
int age = 16;
Person p = new Person(age);
// age is guaranteed to still be 16

public class Person {
  public Person(int initAge) {
    // initAge is a COPY of age
    initAge = 20; // does not alter the value of age in the caller
  }
}
```

Method Parameters: Pass by Value / **Pass by Reference**

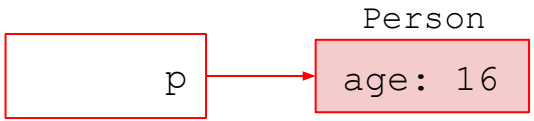
- Object Types / Classes
  - A reference to the Object is passed into the Method
  - The Method can alter the internal value of the Object passed as a parameter; And the caller can see these changes when it access the Object

```
int age = 16;
Person p = new Person(age);
// p.age == 16
Student s = new Student(p);
// p.age == 20

public class Student {
  public Student(Person person) {
    // person is a REFERENCE to the same object in the caller
    person.age = 20; // alters the Person passed in from the the caller
  }
```

Method Parameters: Pass by Value / **Pass by Reference**

- Object Types / Classes
  - A reference to the Object is passed into the Method
  - The Method can alter the internal value of the Object passed as a parameter; And the caller can see these changes when it access the Object

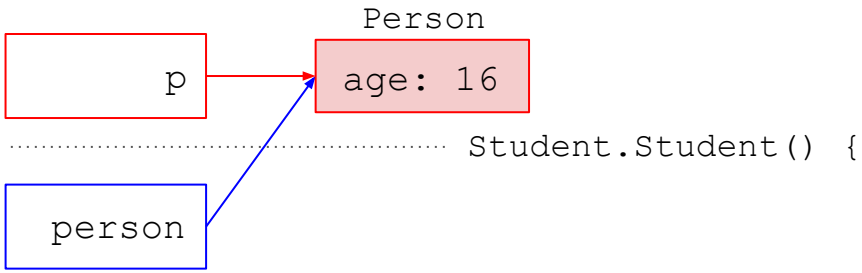
```
int age = 16;
Person p = new Person(age);
// p.age == 16
Student s = new Student(p);
// p.age == 20

public class Student {
  public Student(Person person) {
    // person is a REFERENCE to the same object in the caller
    person.age = 20; // alters the Person passed in from the the caller
  }
```



Method Parameters: Pass by Value / **Pass by Reference**

- Object Types / Classes
  - A reference to the Object is passed into the Method
  - The Method can alter the internal value of the Object passed as a parameter; And the caller can see these changes when it access the Object

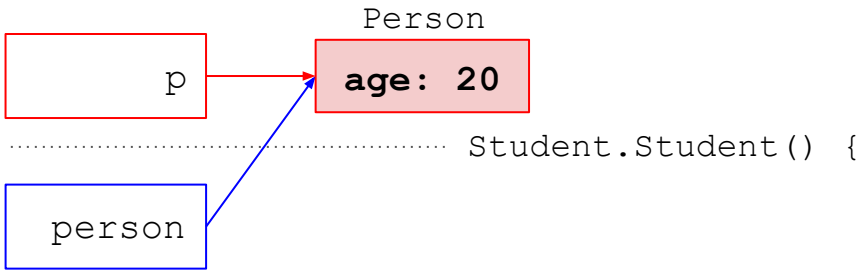
```
int age = 16;
Person p = new Person(age);
// p.age == 16
Student s = new Student(p);
// p.age == 20

public class Student {
  public Student(Person person) {
    // person is a REFERENCE to the same object in the caller
    person.age = 20; // alters the Person passed in from the the caller
  }
```



Method Parameters: Pass by Value / **Pass by Reference**

- Object Types / Classes
  - A reference to the Object is passed into the Method
  - The Method can alter the internal value of the Object passed as a parameter; And the caller can see these changes when it access the Object

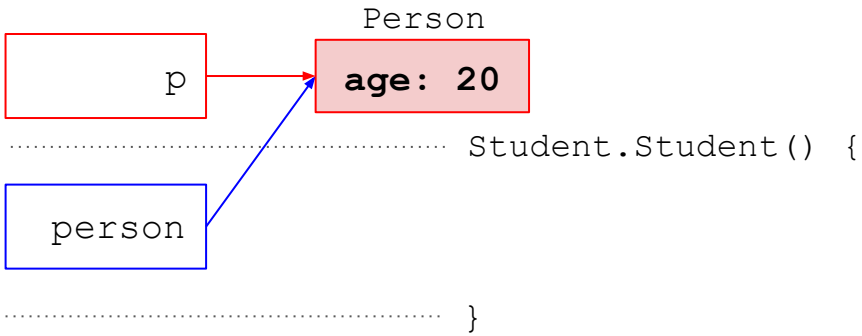
```
int age = 16;
Person p = new Person(age);
// p.age == 16
Student s = new Student(p);
// p.age == 20

public class Student {
  public Student(Person person) {
    // person is a REFERENCE to the same object in the caller
    person.age = 20; // alters the Person passed in from the the caller
  }
```



Method Parameters: Pass by Value / **Pass by Reference**

- Object Types / Classes
  - A reference to the Object is passed into the Method
  - The Method can alter the internal value of the Object passed as a parameter; And the caller can see these changes when it access the Object

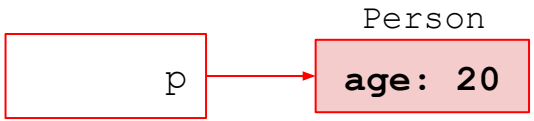
```
int age = 16;
Person p = new Person(age);
// p.age == 16
Student s = new Student(p);
// p.age == 20

public class Student {
  public Student(Person person) {
    // person is a REFERENCE to the same object in the caller
    person.age = 20; // alters the Person passed in from the the caller
  }
```



Method Parameters: Pass by Value / **Pass by Reference**

- Object Types / Classes
  - A reference to the Object is passed into the Method
  - The Method can alter the internal value of the Object passed as a parameter; And the caller can see these changes when it access the Object

```
int age = 16;
Person p = new Person(age);
// p.age == 16
Student s = new Student(p);
// p.age == 20

public class Student {
  public Student(Person person) {
    // person is a REFERENCE to the same object in the caller
    person.age = 20; // alters the Person passed in from the the caller
  }
```



Method Parameters: Pass by Value / **Pass by Reference**

- Object Types / Classes
  - A reference to the Object is passed into the Method
  - The Method can alter the value of the variable (passed as a parameter) that exists in the caller of the Method - **RARELY USED / NOT A BEST PRACTICE**

```
Person p = new Person();
p.age = 16;
Student s = new Student(p);
// p.age now equals 20 -  LIKELY UNEXPECTED BEHAVIOR

public class Student {
  public Student(Person person) {
    // person is a REFERENCE to the same object in the caller
    person.age = 20; // alters the Person passed in from the the caller
  }
}
```

Practice on your own

- CSAwesome 5.6 - Writing Methods
- 5.6.2. Programming Challenge : Song with Parameters
- 5.6.3. Design a Class for your Community
- MusicCollection on Replit