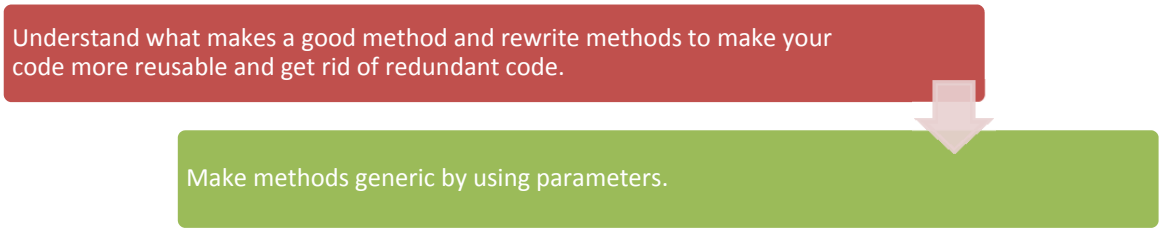
Module III: Good Methods

Learning Objectives



Writing Good Methods

By now you should be able to use methods to accomplish a lot of specific tasks, but these methods really aren't very general. Any time we want to do something different with them, like copy our image to a different location, we have to modify the code and recompile it. If we want to keep the old version we also have to create a new method, with a new name, and copy our new code into it.

To avoid issues like this, we'll make a few general guidelines for what makes a "good" method:

- A method should do one thing, and do it well. The name of the method should tell you what task it does.
- We should avoid copying code between methods. Instead we should take pieces of common code and put it into a new method called by all methods using that code. We sometimes call these *helper methods*. Any time you are rewriting the same code over and over again, it makes sense to put it in a helper method. That way you can write (and debug) that code once, and use it many times.
- A method should have one clear and do-able goal. To achieve that goal, it can (and should) call other methods as needed. If you find that your method is doing many different things, break it up into more than one method.
- A method should be in the class that contains the data it is working with, unless there is a good reason to put it somewhere else.

The last few methods we wrote were *not* general. For example, we have to modify copyPictureOffset () for each place that we might want to put the object, and we need to make a new copy of the method for each new picture we want to copy (even the name is specific to the task the method executes in the current implementation). Instead, we can give the method parameters, which tell it the picture to copy from, as well as the start and end positions for the source and target images. To make our methods apply to all images, in this case we can basically replace our hard-coded values with parameters.

An algorithm for copying pixels from a passed source picture that is more general might look like this:

- Giving a start x and y and end x and y for the source picture
    - If the start x and y and end x and y cover the entire picture then the whole picture will be copied
    - If the start x and y and end x and y are part of the picture then cropping will occur
- Copy the defined region of the source picture to the current picture object with a target start x and target start y
    - If the start x and y are 0 then it copies to the upper left corner
- Loop through the x values between xStart and xEnd
    - Loop through the y values between yStart and yStart
        - Get the pixel from the source picture for the current x and y values
            - Get the pixel from the target picture for the targetStartX + x and targetStartY + y values
            - Set the color in the target pixel to the color in the source pixel

Here's a generic copy method that allows us to copy any section of a source image to a location in a target image.

```
public void copy(Picture sourcePicture, int startX, int startY,
```

```
    int endX, int endY, int targetStartX, int targetStartY) {
  // loop through the x values
  for (int x = startX, tx = targetStartX; x < endX; x++, tx++) {
    // loop through the y values
    for (int y = startY, ty = targetStartY; y < endY; y++, ty++) {
      // copy the source color to the target color
      Pixel sourcePixel = sourcePicture.getPixel(x, y);
      Pixel targetPixel = this.getPixel(tx, ty);
      targetPixel.setColor(sourcePixel.getColor());
    }
  }
}
```

Note... Note that the for loops in the sample code above are written with the initialization, continuation test, and so on all on one line. This doesn't change how they work: It's a stylistic choice, and you may decide which way to do it in part based on how much you have to fit on one line (if there's a lot, you may want to break it up as we have in past examples, if there's not, you can keep it like this).

Now if we want to copy a whole image or any part of an image to another image, all we have to do is call copy. For example, if we have a picture target and a picture source, and we want to copy the rectangle between (0,0) and (70,70) of source to (375,350) of target, we would do target.copy(source, 0,0,70,70,375,350);.

Summary

- To copy one part of a picture to another, we have to control the start and end X and Y values in our source picture as well as the target X and Y values in the target image.
- A method should aim to be as generic as possible, and reduce the amount of redundant code and code copied from other methods. At the same time, it has one clear and do-able goal.
- We can use parameters to make methods reusable.
- If you're copying lots of code from one method to another, *you're doing it wrong*. Instead, create a *helper* method to implement common behavior (like the copy method we made here), and call it from other methods that do more specific things.
- A method should be in the class that contains the data it is working with, unless there is a good reason to put it somewhere else.